



Entwicklerhandbuch für SDK Version 3

AWS SDK für JavaScript



AWS SDK für JavaScript: Entwicklerhandbuch für SDK Version 3

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Die Handelsmarken und Handelsaufmachung von Amazon dürfen nicht in einer Weise in Verbindung mit nicht von Amazon stammenden Produkten oder Services verwendet werden, durch die Kunden irregeführt werden könnten oder Amazon in schlechtem Licht dargestellt oder diskreditiert werden könnte. Alle anderen Handelsmarken, die nicht Eigentum von Amazon sind, gehören den jeweiligen Besitzern, die möglicherweise zu Amazon gehören oder nicht, mit Amazon verbunden sind oder von Amazon gesponsert werden.

Table of Contents

.....	xii
Was ist der AWS SDK für JavaScript?	1
Fangen Sie mit dem SDK an	1
Wartung und Support für SDK-Hauptversionen	2
Verwenden des SDKs mit Node.js	2
Verwenden des SDK mit AWS Amplify	2
Verwenden des SDK mit Webbrowsern	3
Verwendung von Browsern in V3	3
Häufige Anwendungsfälle	4
Informationen zu den Beispielen	4
Ressourcen	4
Erste Schritte	6
SDK-Authentifizierung mit AWS	6
Starten Sie eine AWS Access-Portal-Sitzung	7
Mithilfe der Anmeldeinformationen für die Konsole	8
Weitere Authentifizierungsinformationen	9
Beginnen Sie mit Node.js	9
Das Szenario	9
Voraussetzungen	10
Schritt 1: Richten Sie die Paketstruktur ein und installieren Sie die Client-Pakete	10
Schritt 2: Fügen Sie die erforderlichen Importe und den SDK-Code hinzu	11
Schritt 3: Führen Sie das Beispiel aus	13
Fangen Sie im Browser an	14
Das Szenario	14
Schritt 1: Erstellen Sie einen Amazon Cognito Cognito-Identitätspool und eine IAM-Rolle	15
Schritt 2: Fügen Sie der erstellten IAM-Rolle eine Richtlinie hinzu	16
Schritt 3: Fügen Sie einen Amazon S3 S3-Bucket und ein Objekt hinzu	16
Schritt 4: Richten Sie den Browsercode ein	18
Schritt 5: Führen Sie das Beispiel aus	19
Bereinigen	19
Erste Schritte in React Native	19
Das Szenario	20
Erforderliche Aufgaben	20
Schritt 1: Erstellen Sie einen Amazon Cognito Cognito-Identitätspool	21

Schritt 2: Fügen Sie der erstellten IAM-Rolle eine Richtlinie hinzu	22
Schritt 3: App erstellen mit create-react-native-app	23
Schritt 4: Installieren Sie das Amazon S3 S3-Paket und andere Abhängigkeiten	23
Schritt 5: Schreiben Sie den React Native-Code	24
Schritt 6: Führen Sie das Beispiel aus	27
Mögliche Erweiterungen	29
Richten Sie das SDK ein für JavaScript	30
Voraussetzungen	30
Richten Sie eine AWS Node.js -Umgebung ein	30
Unterstützte Webbrowser	31
Das SDK installieren	32
Laden Sie das SDK	33
Konfigurieren Sie das SDK für JavaScript	34
Konfiguration pro Dienst	34
Legen Sie die Konfiguration pro Dienst fest	35
Legen Sie die Region fest AWS	35
In einem Client-Klassenkonstruktor	36
Verwenden Sie eine Umgebungsvariable	36
Verwenden Sie eine gemeinsam genutzte Konfigurationsdatei	36
Reihenfolge der Festlegung der Region	37
Legen Sie die Anmeldeinformationen fest	37
Bewährte Methoden für Anmeldeinformationen	37
Legen Sie die Anmeldeinformationen in Node.js fest	38
Anmeldeinformationen in einem Webbrowser einrichten	42
Überlegungen zu Node.js	46
Verwenden Sie die integrierten Node.js -Module	46
Verwenden Sie npm-Pakete	47
MaxSockets in Node.js konfigurieren	47
Verbindungen mit Keep-Alive in Node.js wiederverwenden	48
Konfigurieren Sie Proxys für Node.js	49
Registrieren Sie Zertifikatspakete in Node.js	50
Überlegungen zum Browser-Skript	50
Erstellen Sie das SDK für Browser	51
Cross-Origin Resource Sharing (CORS)	51
Mit Webpack bündeln	55
Arbeiten Sie mit AWS Diensten	60

Serviceobjekte erstellen und aufrufen	61
Geben Sie die Parameter des Serviceobjekts an	61
Generierte Clients mit @smithy /types	61
Rufen Sie Dienste asynchron auf	64
Asynchrone Aufrufe verwalten	65
Verwenden Sie async/await	66
Nutze Versprechen	67
Verwenden Sie eine Rückruffunktion	68
Erstellen Sie Service-Client-Anfragen	69
Bearbeiten Sie die Antworten der Servicekunden	71
In der Antwort zurückgegebene Zugriffsdaten	71
Auf Fehlerinformationen zugreifen	71
Arbeiten Sie mit JSON	71
JSON als Serviceobjektparameter	72
Aufrufe protokollieren AWS SDK für JavaScript	73
Verwenden von Middleware zum Protokollieren von Anfragen	74
Verwenden Sie AWS kontobasierte Endpunkte mit DynamoDB	75
Amazon S3 S3-Prüfsummen	75
Hochladen eines Objekts	76
Teilmenge der Codebeispiele mit Anleitung	78
JavaScript ES6/CommonJs-Syntax	80
AWS Elemental MediaConvert Beispiele	82
AWS Lambda Beispiele	100
Amazon-Lex-Beispiele	100
Beispiele für Amazon Polly	100
Beispiele für Amazon Redshift	104
Amazon-SES-Beispiele	112
Amazon SNS-Beispiele	141
Beispiele für Amazon Transcribe	177
Serviceübergreifend: Node.js auf einer Amazon EC2 EC2-Instance einrichten	189
Serviceübergreifend: Amazon API Gateway und Lambda	191
Serviceübergreifend: Geplante Lambda-Ereignisse	207
Serviceübergreifend: Beispiel für Amazon Lex	219
Codebeispiele	233
API Gateway	235
Szenarien	235

Aurora	237
Szenarien	235
Auto Scaling	238
Aktionen	238
Szenarien	235
Amazon Bedrock	280
Erste Schritte	281
Aktionen	238
Amazon Bedrock Runtime	285
Erste Schritte	281
Szenarien	235
Amazon Nova	299
Amazon Nova Canvas	316
Anthropic Claude	319
Cohere Command	329
Meta Llama	333
Mistral AI	339
Agenten für Amazon Bedrock	344
Erste Schritte	281
Aktionen	238
Runtime der Agenten für Amazon Bedrock	358
Aktionen	238
CloudWatch	363
Aktionen	238
CloudWatch Events	372
Aktionen	238
CloudWatch Logs	376
Aktionen	238
Szenarien	235
CodeBuild	393
Aktionen	238
Amazon Cognito Identity	396
Szenarien	235
Amazon Cognito Identity Provider	397
Erste Schritte	281
Aktionen	238

Szenarien	235
Amazon Comprehend	438
Szenarien	235
Amazon DocumentDB	443
Serverless-Beispiele	444
DynamoDB	445
Erste Schritte	281
Grundlagen	447
Aktionen	238
Szenarien	235
Serverless-Beispiele	444
Amazon EC2	638
Erste Schritte	281
Grundlagen	447
Aktionen	238
Szenarien	235
Elastic Load Balancing – Version 2	736
Erste Schritte	281
Aktionen	238
Szenarien	235
AWS Entity Resolution	785
Erste Schritte	281
Aktionen	238
EventBridge	798
Aktionen	238
Szenarien	235
Amazon Glacier	804
Aktionen	238
AWS Glue	807
Erste Schritte	281
Grundlagen	447
Aktionen	238
HealthImaging	832
Erste Schritte	281
Aktionen	238
Szenarien	235

IAM	895
Erste Schritte	281
Grundlagen	447
Aktionen	238
Szenarien	235
AWS IoT SiteWise	988
Erste Schritte	281
Grundlagen	447
Aktionen	238
Kinesis	1019
Aktionen	238
Serverless-Beispiele	444
Lambda	1026
Erste Schritte	281
Grundlagen	447
Aktionen	238
Szenarien	235
Serverless-Beispiele	444
Amazon Lex	1082
Szenarien	235
Amazon Location	1083
Erste Schritte	281
Grundlagen	447
Aktionen	238
Amazon MSK	1115
Serverless-Beispiele	444
Amazon Personalize	1117
Aktionen	238
Amazon Personalize Events	1133
Aktionen	238
Amazon Personalize Runtime	1137
Aktionen	238
Amazon Pinpoint	1141
Aktionen	238
Amazon Polly	1146
Szenarien	235

Amazon RDS	1151
Szenarien	235
Serverless-Beispiele	444
Amazon RDS Data Service	1155
Szenarien	235
Amazon Redshift	1157
Aktionen	238
Amazon Rekognition	1162
Szenarien	235
Amazon S3	1164
Erste Schritte	281
Grundlagen	447
Aktionen	238
Szenarien	235
Serverless-Beispiele	444
SageMaker KI	1298
Erste Schritte	281
Aktionen	238
Szenarien	235
Secrets Manager	1336
Aktionen	238
Amazon SES	1338
Aktionen	238
Szenarien	235
Amazon SNS	1364
Erste Schritte	281
Aktionen	238
Szenarien	235
Serverless-Beispiele	444
Amazon SQS	1405
Erste Schritte	281
Aktionen	238
Szenarien	235
Serverless-Beispiele	444
Step Functions	1436
Aktionen	238

AWS STS	1438
Aktionen	238
Support	1439
Erste Schritte	281
Grundlagen	447
Aktionen	238
Systems Manager	1457
Erste Schritte	281
Grundlagen	447
Aktionen	238
Amazon Textract	1485
Szenarien	235
Amazon Transcribe	1490
Aktionen	238
Szenarien	235
Amazon Translate	1499
Szenarien	235
Sicherheit	1506
Datenschutz	1507
Identitäts- und Zugriffsverwaltung	1508
Zielgruppe	1508
Authentifizierung mit Identitäten	1509
Verwalten des Zugriffs mit Richtlinien	1510
Wie AWS-Services arbeiten Sie mit IAM	1512
Fehlerbehebung bei AWS Identität und Zugriff	1513
Compliance-Validierung	1515
Ausfallsicherheit	1515
Infrastruktursicherheit	1516
Erzwingen Sie eine TLS-Mindestversion	1517
Überprüfen und Erzwingen von TLS in Node.js	1517
Überprüfen und Erzwingen von TLS in einem Browserskript	1520
Die TLS-Version wird in AWS SDK für JavaScript v3-Anfragen abgerufen	1522
Migrieren Sie zu v3	1523
Migrieren Sie mit Codemod zu v3	1523
Verwenden Sie Codemod, um vorhandenen v2-Code zu migrieren	1523
Was ist neu in Version 3	1524

Modularisierte Pakete	1525
Codegröße vergleichen	1526
Befehle in Version 3 aufrufen	1527
Neuer Middleware-Stack	1529
Was ist der Unterschied zwischen der Version 2 und der Version 3	1530
Konstrukteure des Kunden	1531
Anbieter von Anmeldeinformationen	1536
Überlegungen zu Amazon S3	1543
DynamoDB-Dokumentenclient	1544
Kellner und Unterzeichner	1546
Hinweise zu bestimmten Servicekunden	1547
Zusätzliche Unterlagen	1550
Dokumentverlauf	1552
Dokumentverlauf	1552

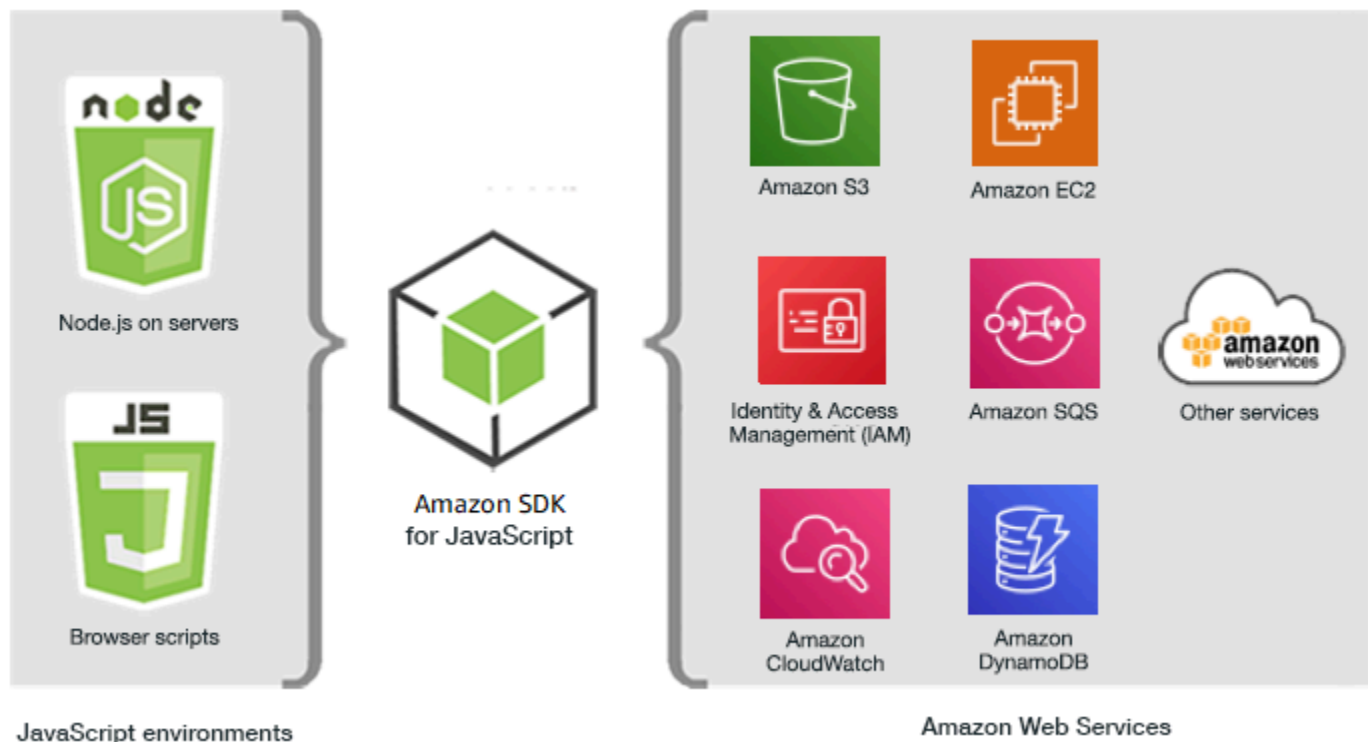
Das [AWS SDK für JavaScript V3-API-Referenzhandbuch](#) beschreibt detailliert alle API-Operationen für die AWS SDK für JavaScript Version 3 (V3).

Die vorliegende Übersetzung wurde maschinell erstellt. Im Falle eines Konflikts oder eines Widerspruchs zwischen dieser übersetzten Fassung und der englischen Fassung (einschließlich infolge von Verzögerungen bei der Übersetzung) ist die englische Fassung maßgeblich.

Was ist der AWS SDK für JavaScript?

Willkommen im AWS SDK für JavaScript Entwicklerhandbuch. Dieses Handbuch enthält allgemeine Informationen zur Einrichtung und Konfiguration von AWS SDK für JavaScript. Es führt Sie auch durch Beispiele und Anleitungen zum Ausführen verschiedener AWS Dienste mit dem AWS SDK für JavaScript.

Das [AWS SDK für JavaScript v3-API-Referenzhandbuch](#) enthält eine JavaScript API für AWS Dienste. Sie können die JavaScript API verwenden, um Bibliotheken oder Anwendungen für [Node.js](#) oder den Browser zu erstellen.



Fangen Sie mit dem SDK an

Wenn Sie bereit sind, das SDK in der Praxis auszuprobieren, folgen Sie den Beispielen unter [Erste Schritte](#).

Informationen zum Einrichten Ihrer Entwicklungsumgebung finden Sie unter [Richten Sie das SDK ein für JavaScript](#).

Wenn Sie derzeit Version 2.x des SDK für verwenden JavaScript, finden Sie spezifische Anleitungen unter [Migration zu](#) Version 3.

Wenn Sie nach Codebeispielen für suchen AWS-Services, finden Sie unter [SDK für JavaScript \(v3\) - Codebeispiele](#).

Wartung und Support für SDK-Hauptversionen

Informationen zur Wartung und zum Support für SDK-Hauptversionen und die ihnen zugrunde liegenden Abhängigkeiten finden Sie im Referenzhandbuch [AWS SDKs und im Tools-Referenzhandbuch](#):

- [AWS SDKs Richtlinien zur Wartung von Tools](#)
- [AWS SDKs und Matrix zur Unterstützung der Tools-Versionen](#)

Verwenden des SDKs mit Node.js

Node.js ist eine plattformübergreifende Laufzeit zum Ausführen serverseitiger Anwendungen JavaScript. Sie können Node.js auf einer Amazon Elastic Compute Cloud (Amazon EC2) -Instance einrichten, um sie auf einem Server auszuführen. Sie können Node.js auch verwenden, um AWS Lambda On-Demand-Funktionen zu schreiben.

Die Verwendung des SDK für Node.js unterscheidet sich von der Art und Weise, wie Sie es JavaScript in einem Webbrowser verwenden. Der Unterschied hängt davon ab, wie das SDK geladen wird und wie die erforderlichen Anmeldeinformationen für den Zugriff auf bestimmte Web-Services abgerufen werden. Wenn sich die Verwendung bestimmter APIs Informationen zwischen Node.js und dem Browser unterscheidet, weisen wir auf diese Unterschiede hin.

Verwenden des SDK mit AWS Amplify

Für browserbasierte Web-, Mobil- und Hybrid-Apps können Sie die [AWS Amplify Bibliothek auch auf GitHub](#) verwenden. Es erweitert das SDK für JavaScript und bietet eine deklarative Schnittstelle.

Note

Frameworks wie Amplify bieten möglicherweise nicht die gleiche Browserunterstützung wie das SDK für JavaScript. Einzelheiten finden Sie in der Dokumentation des Frameworks.

Verwenden des SDK mit Webbrowsern

Alle gängigen Webbrowser unterstützen die Ausführung von JavaScript. JavaScript Code, der in einem Webbrowser ausgeführt wird, wird oft als JavaScriptclientseitig bezeichnet.

Eine Liste der Browser, die von der unterstützt werden AWS SDK für JavaScript, finden Sie unter [Unterstützte Webbrowser](#)

Die Verwendung des SDK für JavaScript in einem Webbrowser unterscheidet sich von der Art und Weise, wie Sie es für Node.js verwenden. Der Unterschied hängt davon ab, wie das SDK geladen wird und wie die erforderlichen Anmeldeinformationen für den Zugriff auf bestimmte Web-Services abgerufen werden. Wenn sich die Verwendung bestimmter APIs Daten zwischen Node.js und dem Browser unterscheidet, weisen wir auf diese Unterschiede hin.

Verwendung von Browsern in V3

V3 ermöglicht es Ihnen, nur das SDK für JavaScript Dateien, die Sie benötigen, zu bündeln und in den Browser aufzunehmen, wodurch der Overhead reduziert wird.

Um Version 3 des SDK for JavaScript in Ihren HTML-Seiten zu verwenden, müssen Sie die erforderlichen Client-Module und alle erforderlichen JavaScript Funktionen mithilfe von Webpack in einer einzigen JavaScript Datei bündeln und diese in einem Skript-Tag auf Ihren HTML-Seiten hinzufügen. <head> Zum Beispiel:

```
<script src="./main.js"></script>
```

Note

Weitere Informationen zu Webpack finden Sie unter [Bündeln Sie Anwendungen mit Webpack](#)

Um Version 2 des SDK für zu verwenden JavaScript, fügen Sie stattdessen ein Skript-Tag hinzu, das auf die neueste Version des V2-SDK verweist. Weitere Informationen finden Sie im [Beispiel](#) im AWS SDK für JavaScript Developer Guide v2.

Häufige Anwendungsfälle

Die Verwendung des SDK für JavaScript in Browserskripten ermöglicht die Realisierung einer Reihe überzeugender Anwendungsfälle. Im Folgenden finden Sie einige Ideen für Dinge, die Sie in einer Browseranwendung erstellen können, indem Sie das SDK für den JavaScript Zugriff auf verschiedene Webdienste verwenden.

- Erstellen Sie eine benutzerdefinierte Konsole für AWS Dienste, in der Sie auf Funktionen aus verschiedenen Regionen und Diensten zugreifen und diese kombinieren können, um Ihre Organisations- oder Projektanforderungen bestmöglich zu erfüllen.
- Verwenden Sie Amazon Cognito Identity, um authentifizierten Benutzerzugriff auf Ihre Browseranwendungen und Websites zu ermöglichen, einschließlich der Verwendung der Drittanbieter-Authentifizierung von Facebook und anderen.
- Verwenden Sie Amazon Kinesis, um Klickstreams oder andere Marketingdaten in Echtzeit zu verarbeiten.
- Verwenden Sie Amazon DynamoDB für die serverlose Datenpersistenz, z. B. für individuelle Benutzereinstellungen für Website-Besucher oder Anwendungsbenutzer.
- Wird verwendet AWS Lambda , um proprietäre Logik zu kapseln, die Sie aus Browser-Skripten aufrufen können, ohne Ihr geistiges Eigentum herunterzuladen und den Benutzern zugänglich zu machen.

Informationen zu den Beispielen

Sie können das SDK im [AWS Code Example JavaScript Repository](#) nach Beispielen durchsuchen.

Ressourcen

Zusätzlich zu diesem Handbuch sind die folgenden Online-Ressourcen für SDK für JavaScript Entwickler verfügbar:

- [AWS SDK für JavaScript V3-API-Referenzhandbuch](#)
- [AWS SDKs Referenzhandbuch und Tools-Referenzhandbuch](#): Enthält Einstellungen, Funktionen und andere grundlegende Konzepte, die allen gemeinsam sind. AWS SDKs
- [JavaScript Blog für Entwickler](#)
- [AWS Re:post](#)
- [JavaScript Beispiele in der Codebibliothek AWS](#)

- [AWS Code-Beispiel-Repository](#)
- [Gitter-Kanal](#)
- [Stack-Überlauf](#)
- [Fragen zu Stack Overflow TaggeAWS -sdk-js](#)
- GitHub
 - [SDK-Quelle](#)
 - [Quelle der Dokumentation](#)

Erste Schritte mit dem AWS SDK für JavaScript

Das AWS SDK für JavaScript bietet Zugriff auf Webdienste in einer Browser- oder Node.js - Umgebung. In diesem Abschnitt finden Sie Übungen für die ersten Schritte, die Ihnen zeigen, wie Sie mit dem AWS SDK für JavaScript jede dieser JavaScript Umgebungen arbeiten können.

Themen

- [SDK-Authentifizierung mit AWS](#)
- [Beginnen Sie mit Node.js](#)
- [Fangen Sie im Browser an](#)
- [Erste Schritte in React Native](#)

SDK-Authentifizierung mit AWS

Sie müssen festlegen, wie sich Ihr Code AWS bei der Entwicklung mit AWS-Services authentifiziert. Sie können den programmatischen Zugriff auf AWS Ressourcen je nach Umgebung und verfügbarem AWS Zugriff auf unterschiedliche Weise konfigurieren.

Informationen zur Auswahl Ihrer Authentifizierungsmethode und deren Konfiguration für das SDK finden Sie unter [Authentifizierung und Zugriff](#) im AWS SDKs Referenzhandbuch zu Tools.

Wir empfehlen neuen Benutzern, die sich lokal weiterentwickeln und von ihrem Arbeitgeber keine Authentifizierungsmethode zur Einrichtung erhalten AWS IAM Identity Center. Diese Methode beinhaltet die Installation von, AWS CLI um die Konfiguration zu vereinfachen und sich regelmäßig beim AWS Zugangsportal anzumelden. Wenn Sie sich für diese Methode entscheiden, sollte Ihre Umgebung die folgenden Elemente enthalten, nachdem Sie das Verfahren für die [IAM Identity Center-Authentifizierung](#) im Referenzhandbuch AWS SDKs und im Tools-Referenzhandbuch abgeschlossen haben:

- Die AWS CLI, mit der Sie eine AWS Access-Portal-Sitzung starten, bevor Sie Ihre Anwendung ausführen.
- Eine [gemeinsam genutzte AWSconfig Datei](#) mit einem [default] Profil mit einer Reihe von Konfigurationswerten, auf die vom SDK aus verwiesen werden kann. Informationen zum Speicherort dieser Datei finden Sie unter [Speicherort der gemeinsam genutzten Dateien](#) im Referenzhandbuch AWS SDKs und im Tools-Referenzhandbuch.

- Die gemeinsam genutzte config Datei legt die [region](#)Einstellung fest. Dies legt die Standardeinstellung AWS-Region fest, die das SDK für AWS Anfragen verwendet. Diese Region wird für SDK-Dienstanforderungen verwendet, für die keine zu verwendende Region angegeben ist.
- Das SDK verwendet die [Konfiguration des SSO-Token-Anbieters](#) des Profils, um Anmeldeinformationen abzurufen, bevor Anfragen an gesendet AWS werden. Der `sso_role_name` Wert, bei dem es sich um eine IAM-Rolle handelt, die mit einem IAM Identity Center-Berechtigungssatz verbunden ist, ermöglicht den Zugriff auf die in Ihrer AWS-Services Anwendung verwendeten.

Die folgende config Beispieldatei zeigt ein Standardprofil, das mit der Konfiguration des SSO-Token-Anbieters eingerichtet wurde. Die `sso_session` Einstellung des Profils bezieht sich auf den benannten [sso-sessionAbschnitt](#). Der `sso-session` Abschnitt enthält Einstellungen zum Initiieren einer AWS Access-Portal-Sitzung.

```
[default]
sso_session = my-sso
sso_account_id = 111122223333
sso_role_name = SampleRole
region = us-east-1
output = json

[sso-session my-sso]
sso_region = us-east-1
sso_start_url = https://provided-domain.awsapps.com/start
sso_registration_scopes = sso:account:access
```

Für das AWS SDK für JavaScript Version 3 müssen Ihrer Anwendung keine zusätzlichen Pakete (wie SSO undSSO0IDC) hinzugefügt werden, um die IAM Identity Center-Authentifizierung verwenden zu können.

Einzelheiten zur expliziten Verwendung dieses Anmeldeinformationsanbieters finden Sie [fromSSO\(\)](#) auf der npm-Website (Node.js package manager).

Starten Sie eine AWS Access-Portal-Sitzung

Bevor Sie eine Zugriffsanwendung ausführen AWS-Services, benötigen Sie eine aktive AWS Access-Portal-Sitzung, damit das SDK die IAM Identity Center-Authentifizierung zur Auflösung von Anmeldeinformationen verwenden kann. Abhängig von Ihrer konfigurierten Sitzungsdauer läuft Ihr

Zugriff irgendwann ab und das SDK wird auf einen Authentifizierungsfehler stoßen. Um sich beim AWS Zugriffsportal anzumelden, führen Sie den folgenden Befehl in der aus AWS CLI.

```
aws sso login
```

Wenn Sie die Anweisungen befolgt haben und ein Standardprofil eingerichtet haben, müssen Sie den Befehl nicht mit einer `--profile` Option aufrufen. Wenn die Konfiguration Ihres SSO-Token-Anbieters ein benanntes Profil verwendet, lautet der Befehl `aws sso login --profile named-profile`.

Führen Sie den folgenden AWS CLI Befehl aus, um optional zu testen, ob Sie bereits eine aktive Sitzung haben.

```
aws sts get-caller-identity
```

Wenn Ihre Sitzung aktiv ist, werden in der Antwort auf diesen Befehl das in der gemeinsam genutzten config Datei konfigurierte IAM Identity Center-Konto und der Berechtigungssatz gemeldet.

Note

Wenn Sie bereits eine aktive AWS Access-Portal-Sitzung haben und ausführen `aws sso login`, müssen Sie keine Anmeldeinformationen angeben.

Beim Anmeldevorgang werden Sie möglicherweise aufgefordert, den AWS CLI Zugriff auf Ihre Daten zu gewähren. Da AWS CLI das auf dem SDK für Python aufbaut, können Berechtigungsnachrichten Variationen des `botocore` Namens enthalten.

Mithilfe der Anmeldeinformationen für die Konsole

Sie können Ihre vorhandenen Anmeldedaten für die AWS Management Console für den programmgesteuerten Zugriff AWS auf Dienste verwenden. AWS Generiert nach einem browserbasierten Authentifizierungsablauf temporäre Anmeldeinformationen, die in allen lokalen Entwicklungstools wie der AWS CLI und dem AWS SDK für JavaScript funktionieren. Diese Funktion vereinfacht die Konfiguration und Verwaltung von AWS CLI-Anmeldeinformationen. Folgen Sie den Anweisungen zur [Anmeldung für die AWS lokale Entwicklung mit Konsolenanmeldedaten](#), um zu erfahren, wie Sie loslegen können.

Wenn Sie den `aws login` Befehl ausführen, können Sie aus Ihren aktiven Konsolensitzungen auswählen oder sich über den browserbasierten Authentifizierungsablauf anmelden. Dabei

werden automatisch temporäre Anmeldeinformationen generiert. Das AWS SDK für JavaScript aktualisiert die Anmeldeinformationen automatisch 5 Minuten vor Ablauf, wobei jeder Satz von Anmeldeinformationen bis zu 12 Stunden gültig ist. Weitere Informationen finden Sie unter [fromLoginCredentials\(\)](#).

Weitere Authentifizierungsinformationen

Menschliche Benutzer, auch bekannt als menschliche Identitäten, sind die Personen, Administratoren, Entwickler, Betreiber und Verbraucher Ihrer Anwendungen. Sie müssen über eine Identität verfügen, um auf Ihre AWS Umgebungen und Anwendungen zugreifen zu können. Menschliche Benutzer, die Mitglieder Ihres Unternehmens sind — also Sie, der Entwickler — werden als Personalidentitäten bezeichnet.

Verwenden Sie beim Zugriff AWS temporäre Anmeldeinformationen. Sie können einen Identitätsanbieter für Ihre menschlichen Benutzer verwenden, um Verbundzugriff auf AWS Konten zu ermöglichen, indem Sie Rollen übernehmen, die temporäre Anmeldeinformationen bereitstellen. Für eine zentralisierte Zugriffsverwaltung empfehlen wir Ihnen, AWS IAM Identity Center (IAM Identity Center) zu verwenden, um den Zugriff auf Ihre Konten und die Berechtigungen innerhalb dieser Konten zu verwalten. Weitere Alternativen finden Sie im Folgenden:

- Weitere Informationen zu bewährten Methoden finden Sie unter [Bewährte Methoden für die Sicherheit in IAM](#) im IAM-Benutzerhandbuch.
- Informationen zum Erstellen kurzfristiger AWS Anmeldeinformationen finden Sie unter [Temporäre Sicherheitsanmeldedaten](#) im IAM-Benutzerhandbuch.
- Weitere Informationen zu anderen AWS SDKs für JavaScript V3-Anmeldeinformationsanbieter finden Sie unter [Standardisierte Anmeldeinformationsanbieter](#) im Referenzhandbuch AWS SDKs und im Tools-Referenzhandbuch.

Beginnen Sie mit Node.js

Diese Anleitung zeigt Ihnen, wie Sie ein NPM-Paket initialisieren, Ihrem Paket einen Service-Client hinzufügen und das JavaScript SDK verwenden, um eine Serviceaktion aufzurufen.

Das Szenario

Erstellen Sie ein neues NPM-Paket mit einer Hauptdatei, die Folgendes tut:

- Erstellt einen Amazon Simple Storage Service-Bucket

- Fügt ein Objekt in den Amazon S3 S3-Bucket ein
- Liest das Objekt im Amazon S3 S3-Bucket
- Bestätigt, ob der Benutzer Ressourcen löschen möchte

Voraussetzungen

Bevor Sie das Beispiel ausführen können, müssen Sie Folgendes tun:

- Konfigurieren Sie Ihre SDK-Authentifizierung. Weitere Informationen finden Sie unter [SDK-Authentifizierung mit AWS](#).
- Installieren Sie [Node.js](#). AWS empfiehlt, die Active LTS-Version von Node.js für die Entwicklung zu verwenden.

Schritt 1: Richten Sie die Paketstruktur ein und installieren Sie die Client-Pakete

So richten Sie die Paketstruktur ein und installieren die Client-Pakete:

1. Erstellen Sie einen neuen Ordner `nodegetstarted`, der das Paket enthalten soll.
2. Navigieren Sie von der Befehlszeile aus zu dem neuen Ordner.
3. Führen Sie den folgenden Befehl aus, um eine `package.json` Standarddatei zu erstellen:

```
npm init -y
```

4. Führen Sie den folgenden Befehl aus, um das Amazon S3 S3-Client-Paket zu installieren:

```
npm i @aws-sdk/client-s3
```

5. `"type": "module"` Zur `package.json` Datei hinzufügen. Dadurch wird Node.js angewiesen, die moderne ESM-Syntax zu verwenden. Das finale `package.json` sollte etwa wie folgt aussehen:

```
{  
  "name": "example-javascriptv3-get-started-node",  
  "version": "1.0.0",
```

```
"description": "This guide shows you how to initialize an NPM package, add a
service client to your package, and use the JavaScript SDK to call a service
action.",
"main": "index.js",
"scripts": {
"test": "vitest run **/*.unit.test.js"
},
"author": "Your Name",
"license": "Apache-2.0",
"dependencies": {
"@aws-sdk/client-s3": "^3.420.0"
},
"type": "module"
}
```

Schritt 2: Fügen Sie die erforderlichen Importe und den SDK-Code hinzu

Fügen Sie den folgenden Code zu einer Datei hinzu, die `index.js` im `nodegetstarted` Ordner benannt ist.

```
// This is used for getting user input.
import { createInterface } from "node:readline/promises";

import {
  S3Client,
  PutObjectCommand,
  CreateBucketCommand,
  DeleteObjectCommand,
  DeleteBucketCommand,
  paginateListObjectsV2,
  GetObjectCommand,
} from "@aws-sdk/client-s3";

export async function main() {
  // A region and credentials can be declared explicitly. For example
  // `new S3Client({ region: 'us-east-1', credentials: {...} })` would
  // initialize the client with those settings. However, the SDK will
  // use your local configuration and credentials if those properties
  // are not defined here.
  const s3Client = new S3Client({});
```

```
// Create an Amazon S3 bucket. The epoch timestamp is appended
// to the name to make it unique.
const bucketName = `test-bucket-${Date.now()}`;
await s3Client.send(
  new CreateBucketCommand({
    Bucket: bucketName,
  }),
);

// Put an object into an Amazon S3 bucket.
await s3Client.send(
  new PutObjectCommand({
    Bucket: bucketName,
    Key: "my-first-object.txt",
    Body: "Hello JavaScript SDK!",
  }),
);

// Read the object.
const { Body } = await s3Client.send(
  new GetObjectCommand({
    Bucket: bucketName,
    Key: "my-first-object.txt",
  }),
);

console.log(await Body.transformToString());

// Confirm resource deletion.
const prompt = createInterface({
  input: process.stdin,
  output: process.stdout,
});

const result = await prompt.question("Empty and delete bucket? (y/n) ");
prompt.close();

if (result === "y") {
  // Create an async iterator over lists of objects in a bucket.
  const paginator = paginateListObjectsV2(
    { client: s3Client },
    { Bucket: bucketName },
  );
  for await (const page of paginator) {
```



```
const objects = page.Contents;
if (objects) {
  // For every object in each page, delete it.
  for (const object of objects) {
    await s3Client.send(
      new DeleteObjectCommand({ Bucket: bucketName, Key: object.Key }),
    );
  }
}

// Once all the objects are gone, the bucket can be deleted.
await s3Client.send(new DeleteBucketCommand({ Bucket: bucketName }));
}

// Call a function if this file was run directly. This allows the file
// to be runnable without running on import.
import { fileURLToPath } from "node:url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main();
}
```

Den Beispielcode finden Sie [hier auf GitHub](#).

Schritt 3: Führen Sie das Beispiel aus

Note

Denken Sie daran, sich anzumelden! Wenn Sie IAM Identity Center zur Authentifizierung verwenden, denken Sie daran, sich mit dem AWS CLI `aws sso login` folgenden Befehl anzumelden.

1. Führen Sie `node index.js`.
2. Wählen Sie aus, ob der Bucket geleert und gelöscht werden soll.
3. Wenn Sie den Bucket nicht löschen, stellen Sie sicher, dass Sie ihn manuell leeren und später löschen.

Fangen Sie im Browser an

Dieser Abschnitt führt Sie durch ein Beispiel, das zeigt, wie Sie Version 3 (V3) des AWS SDK für JavaScript im Browser ausführen.

Note

Die Ausführung von V3 im Browser unterscheidet sich geringfügig von Version 2 (V2). Weitere Informationen finden Sie unter [Verwendung von Browsern in V3](#).

Weitere Beispiele für die Verwendung von (V3) des AWS SDK für JavaScript finden Sie unter [SDK für JavaScript \(v3\) -Codebeispiele](#).

Dieses Beispiel für eine Webanwendung zeigt Ihnen:

- So greifen Sie mit Amazon Cognito zur Authentifizierung auf AWS Dienste zu.
- Wie liest man eine Liste von Objekten in einem Amazon Simple Storage Service (Amazon S3) - Bucket mithilfe einer AWS Identity and Access Management (IAM) -Rolle.

Note

Dieses Beispiel wird nicht AWS IAM Identity Center für die Authentifizierung verwendet.

Das Szenario

Amazon S3 ist ein Objektspeicherservice, der branchenführende Skalierbarkeit, Datenverfügbarkeit, Sicherheit und Leistung bietet. Sie können Amazon S3 verwenden, um Daten als Objekte in Containern, sogenannten Buckets, zu speichern. Weitere Informationen zu Amazon S3 finden Sie im [Amazon S3 S3-Benutzerhandbuch](#).

Dieses Beispiel zeigt Ihnen, wie Sie eine Web-App einrichten und ausführen, die eine IAM-Rolle zum Lesen aus einem Amazon S3 S3-Bucket annimmt. Das Beispiel verwendet die React-Frontend-Bibliothek und die Vite-Frontend-Tools, um eine Entwicklungsumgebung bereitzustellen. JavaScript Die Web-App verwendet einen Amazon Cognito Cognito-Identitätspool, um Anmeldeinformationen bereitzustellen, die für den Zugriff auf AWS Dienste erforderlich sind. Das mitgelieferte Codebeispiel

demonstriert die grundlegenden Muster für das Laden und Verwenden des AWS SDK JavaScript in Web-Apps.

Schritt 1: Erstellen Sie einen Amazon Cognito Cognito-Identitätspool und eine IAM-Rolle

In dieser Übung erstellen und verwenden Sie einen Amazon Cognito Cognito-Identitätspool, um nicht authentifizierten Zugriff auf Ihre Web-App für den Amazon S3 S3-Service bereitzustellen. Durch die Erstellung eines Identitätspools wird auch eine AWS Identity and Access Management (IAM-) Rolle zur Unterstützung nicht authentifizierter Gastbenutzer erstellt. In diesem Beispiel werden wir nur mit der Rolle „Nicht authentifizierter Benutzer“ arbeiten, damit wir uns auf die Aufgabe konzentrieren können. Sie können die Unterstützung für einen Identitätsanbieter und authentifizierte Benutzer zu einem späteren Zeitpunkt integrieren. Weitere Informationen zum Hinzufügen eines Amazon Cognito-Identitätspools finden Sie unter [Tutorial: Creating an Identity Pool](#) im Amazon Cognito Developer Guide.

So erstellen Sie einen Amazon Cognito Cognito-Identitätspool und die zugehörige IAM-Rolle

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon Cognito Cognito-Konsole unter <https://console.aws.amazon.com/cognito/>.
2. Wählen Sie im linken Navigationsbereich Identity Pools aus.
3. Wählen Sie Identitätspool erstellen.
4. Wählen Sie unter Vertrauensstellung im Identitätspool konfigurieren die Option Gastzugriff für die Benutzerauthentifizierung aus.
5. Wählen Sie unter Berechtigungen konfigurieren die Option Neue IAM-Rolle erstellen aus und geben Sie einen Namen (z. B. getStartedRole) in den IAM-Rollennamen ein.
6. Geben Sie unter Eigenschaften konfigurieren einen Namen ein (z. B. getStartedPool) im Feld Identitätspoolname.
7. Bestätigen Sie unter Überprüfen und erstellen die Auswahl, die Sie für Ihren neuen Identitätspool getroffen haben. Wählen Sie Bearbeiten, um zum Assistenten zurückzukehren und Einstellungen zu ändern. Wählen Sie danach Identitätspool erstellen aus.
8. Notieren Sie sich die Identitätspool-ID und die Region des neu erstellten Amazon Cognito Cognito-Identitätspools. Sie benötigen diese Werte, um sie zu ersetzen **IDENTITY_POOL_ID** und einzugeben **REGION**. [Schritt 4: Richten Sie den Browsercode ein](#)

Nachdem Sie Ihren Amazon Cognito Cognito-Identitätspool erstellt haben, können Sie Berechtigungen für Amazon S3 hinzufügen, die von Ihrer Web-App benötigt werden.

Schritt 2: Fügen Sie der erstellten IAM-Rolle eine Richtlinie hinzu

Um den Zugriff auf einen Amazon S3 S3-Bucket in Ihrer Web-App zu aktivieren, verwenden Sie die nicht authentifizierte IAM-Rolle (z. B. `getStartedRole`), die für Ihren Amazon Cognito Cognito-Identitätspool erstellt wurde (z. B.). `getStartedPool` Dazu müssen Sie der Rolle eine IAM-Richtlinie hinzufügen. Weitere Informationen zum Ändern von IAM-Rollen finden Sie unter [Ändern einer Rollenberechtigungsrichtlinie](#) im IAM-Benutzerhandbuch.

Um der IAM-Rolle, die nicht authentifizierte Benutzern zugeordnet ist, eine Amazon S3 S3-Richtlinie hinzuzufügen

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die IAM-Konsole unter. <https://console.aws.amazon.com/iam/>
2. Wählen Sie im linken Navigationsbereich Roles aus.
3. Wählen Sie den Namen der Rolle aus, die Sie ändern möchten (z. B. `getStartedRole`), und wählen Sie dann die Registerkarte Berechtigungen.
4. Wählen Sie Berechtigungen hinzufügen aus und wählen Sie dann Richtlinien anfügen aus.
5. Suchen Sie auf der Seite „Berechtigungen hinzufügen“ für diese Rolle nach dem Kontrollkästchen für AmazonS3 und aktivieren Sie es. `ReadOnlyAccess`

Note

Sie können diesen Prozess verwenden, um den Zugriff auf jeden AWS Dienst zu aktivieren.

6. Wählen Sie Add permissions (Berechtigungen hinzufügen) aus.

Nachdem Sie Ihren Amazon Cognito Cognito-Identitätspool erstellt und Ihrer IAM-Rolle für nicht authentifizierte Benutzer Berechtigungen für Amazon S3 hinzugefügt haben, können Sie einen Amazon S3 S3-Bucket hinzufügen und konfigurieren.

Schritt 3: Fügen Sie einen Amazon S3 S3-Bucket und ein Objekt hinzu

In diesem Schritt fügen Sie einen Amazon S3 S3-Bucket und ein Objekt für das Beispiel hinzu. Sie werden auch Cross-Origin Resource Sharing (CORS) für den Bucket aktivieren. Weitere

Informationen zum Erstellen von Amazon S3-Buckets und -Objekten finden Sie unter [Erste Schritte mit Amazon S3](#) im Amazon S3 S3-Benutzerhandbuch.

Um einen Amazon S3 S3-Bucket und ein Objekt mit CORS hinzuzufügen

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Amazon S3 S3-Konsole unter <https://console.aws.amazon.com/s3/>.
2. Wählen Sie im linken Navigationsbereich Buckets und dann Create Bucket aus.
3. Geben Sie einen Bucket-Namen ein, der den [Regeln für die Benennung von Buckets](#) entspricht (z. B. getstartedbucket), und wählen Sie Create Bucket aus.
4. Wählen Sie den Bucket aus, den Sie erstellt haben, und klicken Sie dann auf den Tab Objekte. Klicken Sie anschließend auf Upload.
5. Wählen Sie unter Files and folders (Dateien und Ordner) die Option Add files (Dateien hinzufügen) aus.
6. Wählen Sie eine hochzuladende Datei und dann Öffnen aus. Wählen Sie dann Hochladen, um den Upload des Objekts in Ihren Bucket abzuschließen.
7. Wählen Sie als Nächstes den Tab „Berechtigungen“ Ihres Buckets und anschließend im Abschnitt Cross-Origin Resource Sharing (CORS) die Option Bearbeiten aus. Geben Sie den folgenden JSON-Code ein:

```
[
  {
    "AllowedHeaders": [
      "*"
    ],
    "AllowedMethods": [
      "GET"
    ],
    "AllowedOrigins": [
      "*"
    ],
    "ExposeHeaders": []
  }
]
```

8. Wählen Sie Änderungen speichern aus.

Nachdem Sie einen Amazon S3 S3-Bucket und ein Objekt hinzugefügt haben, können Sie den Browsercode einrichten.

Schritt 4: Richten Sie den Browsercode ein

Die Beispielanwendung besteht aus einer einseitigen React-Anwendung. Die Dateien für dieses Beispiel finden Sie [hier auf GitHub](#).

Um die Beispielanwendung einzurichten

1. Installieren Sie [Node.js](#).
2. Klonen Sie das [AWS Codebeispiel-Repository](#) von der Befehlszeile aus:

```
git clone --depth 1 https://github.com/awsdocs/aws-doc-sdk-examples.git
```

3. Navigieren Sie zur Beispielanwendung:

```
cd aws-doc-sdk-examples/javascriptv3/example_code/web/s3/list-objects/
```

4. Führen Sie den folgenden Befehl aus, um die erforderlichen Pakete zu installieren:

```
npm install
```

5. Öffnen Sie anschließend `src/App.tsx` in einem Texteditor und gehen Sie wie folgt vor:
 - **`YOUR_IDENTITY_POOL_ID`** Ersetzen Sie es durch die Amazon Cognito Cognito-Identitätspool-ID, die Sie notiert [Schritt 1: Erstellen Sie einen Amazon Cognito Cognito-Identitätspool und eine IAM-Rolle](#) haben.
 - Ersetzen Sie den Wert für Region durch die Region, die Ihrem Amazon S3-Bucket und Amazon Cognito Cognito-Identitätspool zugewiesen wurde. Beachten Sie, dass die Regionen für beide Dienste identisch sein müssen (z. B. `us-east-2`).
 - Ersetzen Sie es durch **`bucket-name`** den Bucket-Namen, in dem Sie es erstellt haben. [Schritt 3: Fügen Sie einen Amazon S3 S3-Bucket und ein Objekt hinzu](#)

Nachdem Sie den Text ersetzt haben, speichern Sie die `App.tsx` Datei. Sie sind jetzt bereit, die Web-App auszuführen.

Schritt 5: Führen Sie das Beispiel aus

Um die Beispielanwendung auszuführen

1. Navigieren Sie von der Befehlszeile aus zur Beispielanwendung:

```
cd aws-doc-sdk-examples/javascriptv3/example_code/web/s3/list-objects/
```

2. Führen Sie in der Befehlszeile den folgenden Befehl aus:

```
npm run dev
```

Die Vite-Entwicklungsumgebung wird mit der folgenden Meldung ausgeführt:

```
VITE v4.3.9 ready in 280 ms

# Local:   http://localhost:5173/
# Network: use --host to expose
# press h to show help
```

3. Navigieren Sie in Ihrem Webbrowser zu der oben angegebenen URL (z. B. <http://localhost:5173/>). Die Beispiel-App zeigt Ihnen eine Liste von Objektdatenamen in Ihrem Amazon S3 S3-Bucket.

Bereinigen

Gehen Sie wie folgt vor, um die Ressourcen zu bereinigen, die Sie in diesem Tutorial erstellt haben:

- Löschen Sie in [der Amazon S3 S3-Konsole](#) alle Objekte und erstellten Buckets (z. B. `getstartedbucket`).
- Löschen Sie in [der IAM-Konsole](#) den Rollennamen (z. B.). `getStartedRole`
- Löschen Sie in [der Amazon Cognito Cognito-Konsole](#) den Namen des Identitätspools (z. B. `getStartedPool`).

Erste Schritte in React Native

Dieses Tutorial zeigt Ihnen, wie Sie mit React Native [CLI eine React Native-App](#) erstellen können.



Dieses Tutorial zeigt dir:

- So installieren und integrieren Sie das AWS SDK für Module der JavaScript Version 3 (V3), die Ihr Projekt verwendet.
- So schreiben Sie Code, der eine Verbindung zu Amazon Simple Storage Service (Amazon S3) herstellt, um einen Amazon S3-Bucket zu erstellen und zu löschen.

Das Szenario

Amazon S3 ist ein Cloud-Service, mit dem Sie beliebige Datenmengen zu jeder Zeit und von überall im Internet speichern und abrufen können. React Native ist ein Entwicklungsframework, mit dem Sie mobile Anwendungen erstellen können. Dieses Tutorial zeigt Ihnen, wie Sie eine React Native-App erstellen können, die eine Verbindung zu Amazon S3 herstellt, um einen Amazon S3 S3-Bucket zu erstellen und zu löschen.

Die App verwendet das folgende AWS SDK für JavaScript APIs:

- [CognitoIdentityClient](#) Konstruktor
- [S3](#) Konstruktor

Erforderliche Aufgaben

Note

Wenn Sie bereits einen der folgenden Schritte durch andere Tutorials oder eine vorhandene Konfiguration ausgeführt haben, überspringen Sie diese Schritte.

Dieser Abschnitt enthält die minimale Einrichtung, die erforderlich ist, um dieses Tutorial abzuschließen. Sie sollten dies nicht als vollständiges Setup betrachten. Sehen Sie dazu [Richten Sie das SDK ein für JavaScript](#).

- Installieren Sie die folgenden Tools:

- [npm](#)
- [Node.js](#)
- [Xcode](#), wenn Sie auf iOS testen
- [Android Studio](#), wenn Sie auf Android testen
- Richten Sie Ihre [React Native-Entwicklungsumgebung](#) ein
- Richten Sie die Projektumgebung ein, um diese TypeScript Node-Beispiele auszuführen, und installieren Sie das erforderliche AWS SDK für Module JavaScript und Module von Drittanbietern. Folgen Sie den Anweisungen auf [GitHub](#).
- Sie müssen festlegen, wie sich Ihr Code AWS bei der Entwicklung mit AWS Diensten authentifiziert. Weitere Informationen finden Sie unter [SDK-Authentifizierung mit AWS](#).

 Note

Die IAM-Rolle für dieses Beispiel sollte so eingestellt sein, dass sie FullAccessAmazonS3-Berechtigungen verwendet.

Schritt 1: Erstellen Sie einen Amazon Cognito Cognito-Identitätspool

In dieser Übung erstellen und verwenden Sie einen Amazon Cognito Cognito-Identitätspool, um nicht authentifizierten Zugriff auf Ihre App für den Amazon S3 S3-Service bereitzustellen. Durch die Erstellung eines Identitätspools werden auch zwei AWS Identity and Access Management (IAM-) Rollen erstellt, eine zur Unterstützung von Benutzern, die von einem Identitätsanbieter authentifiziert wurden, und die andere zur Unterstützung nicht authentifizierter Gastbenutzer.

In dieser Übung arbeiten wir nur mit der Rolle für nicht authentifizierte Benutzer, um uns auf die Aufgabe zu konzentrieren. Sie können die Unterstützung für einen Identitätsanbieter und authentifizierte Benutzer zu einem späteren Zeitpunkt integrieren.

So erstellen Sie einen Amazon Cognito Cognito-Identitätspool

1. Melden Sie sich bei der Amazon Cognito-Konsole an AWS-Managementkonsole und öffnen Sie die Amazon Cognito Cognito-Konsole in der [Amazon Web Services Console](#).
2. Wählen Sie auf der Eröffnungsseite der Konsole Identity Pools aus.
3. Wählen Sie auf der nächsten Seite die Option Create new identity pool (Neuen Identitäten-Pool erstellen) aus.

 Note

Wenn es keine anderen Identitätspools gibt, überspringt die Amazon Cognito Cognito-Konsole diese Seite und öffnet stattdessen die nächste Seite.

4. Wählen Sie unter Vertrauensstellung im Identitätspool konfigurieren die Option Gastzugriff für die Benutzerauthentifizierung aus.
5. Wählen Sie unter Berechtigungen konfigurieren die Option Neue IAM-Rolle erstellen aus und geben Sie einen Namen (z. B. `getStartedReactRolle`) in den IAM-Rollennamen ein.
6. Geben Sie unter Eigenschaften konfigurieren einen Namen (z. B. `getStartedReactPool`) im Feld Identitätspoolname ein.
7. Bestätigen Sie unter Überprüfen und erstellen die Auswahl, die Sie für Ihren neuen Identitätspool getroffen haben. Wählen Sie Bearbeiten, um zum Assistenten zurückzukehren und Einstellungen zu ändern. Wählen Sie danach Identitätspool erstellen aus.
8. Notieren Sie sich die Identitätspool-ID und die Region für diesen neu erstellten Identitätspool. Sie benötigen diese Werte als Ersatz für *region* und *identityPoolId* in Ihrem Browser-Skript.

Nachdem Sie Ihren Amazon Cognito Cognito-Identitätspool erstellt haben, können Sie Berechtigungen für Amazon S3 hinzufügen, die von Ihrer React Native-App benötigt werden.

Schritt 2: Fügen Sie der erstellten IAM-Rolle eine Richtlinie hinzu

Um den Browser-Skriptzugriff auf Amazon S3 zum Erstellen und Löschen eines Amazon S3 S3-Buckets zu aktivieren, verwenden Sie die nicht authentifizierte IAM-Rolle, die für Ihren Amazon Cognito Cognito-Identitätspool erstellt wurde. Dazu müssen Sie der Rolle eine IAM-Richtlinie hinzufügen. Weitere Informationen zu IAM-Rollen finden Sie unter [Creating a Role to Delegate Permissions to an AWS Service](#) im IAM-Benutzerhandbuch.

Um der IAM-Rolle, die nicht authentifizierte Benutzern zugeordnet ist, eine Amazon S3 S3-Richtlinie hinzuzufügen

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die IAM-Konsole unter <https://console.aws.amazon.com/iam/>
2. Wählen Sie im linken Navigationsbereich Roles aus.
3. Wählen Sie den Namen der Rolle aus, die Sie ändern möchten (z. B. `getStartedRole`), und wählen Sie dann die Registerkarte Berechtigungen.

4. Wählen Sie Berechtigungen hinzufügen aus und wählen Sie dann Richtlinien anfügen aus.
5. Suchen Sie auf der Seite „Berechtigungen hinzufügen“ für diese Rolle nach dem Kontrollkästchen für AmazonS3 und aktivieren Sie es. ReadOnlyAccess

 Note

Sie können diesen Prozess verwenden, um den Zugriff auf jeden AWS Dienst zu aktivieren.

6. Wählen Sie Add permissions (Berechtigungen hinzufügen) aus.

Nachdem Sie Ihren Amazon Cognito Cognito-Identitätspool erstellt und Ihrer IAM-Rolle für nicht authentifizierte Benutzer Berechtigungen für Amazon S3 hinzugefügt haben, können Sie die App erstellen.

Schritt 3: App erstellen mit create-react-native-app

Erstellen Sie eine React Native-App, indem Sie den folgenden Befehl ausführen.

```
npx react-native init ReactNativeApp --npm
```

Schritt 4: Installieren Sie das Amazon S3 S3-Paket und andere Abhängigkeiten

Führen Sie im Verzeichnis des Projekts die folgenden Befehle aus, um das Amazon S3 S3-Paket zu installieren.

```
npm install @aws-sdk/client-s3
```

Mit diesem Befehl wird das Amazon S3-Paket in Ihrem Projekt installiert und aktualisiertpackage.json, sodass Amazon S3 als Projektabhängigkeit aufgeführt wird. Informationen zu diesem Paket finden Sie, indem Sie auf der <https://www.npmjs.com/> npm-Website nach "@aws-sdk" suchen.

Diese Pakete und der zugehörige Code werden im node_modules-Unterverzeichnis Ihres Projekts installiert.

Weitere Informationen zur Installation von Paketen mit Node.js finden Sie unter [Pakete lokal herunterladen und installieren](#) und [Module der Datei Node.js erstellen](#) auf der [npm-Website \(Node.js package manager\)](#). Informationen zum Herunterladen und Installieren des AWS SDK für finden Sie JavaScript unter [Installieren Sie das SDK für JavaScript](#).

Installieren Sie andere Abhängigkeiten, die für die Authentifizierung erforderlich sind.

```
npm install @aws-sdk/client-cognito-identity @aws-sdk/credential-provider-cognito-identity
```

Schritt 5: Schreiben Sie den React Native-Code

Fügen Sie den folgenden Code zum `hinzuApp.tsx`. Ersetzen Sie *identityPoolId* und *region* durch die Identitätspool-ID und die Region, in der Ihr Amazon S3 S3-Bucket erstellt wird.

```
import React, { useCallback, useState } from "react";
import { Button, StyleSheet, Text, TextInput, View } from "react-native";
import "react-native-get-random-values";
import "react-native-url-polyfill/auto";

import {
  S3Client,
  CreateBucketCommand,
  DeleteBucketCommand,
} from "@aws-sdk/client-s3";
import { fromCognitoIdentityPool } from "@aws-sdk/credential-providers";

const client = new S3Client({
  // The AWS Region where the Amazon Simple Storage Service (Amazon S3) bucket will be
  // created. Replace this with your Region.
  region: "us-east-1",
  credentials: fromCognitoIdentityPool({
    // Replace the value of 'identityPoolId' with the ID of an Amazon Cognito identity
    // pool in your Amazon Cognito Region.
    identityPoolId: "us-east-1:edbe2c04-7f5d-469b-85e5-98096bd75492",
    // Replace the value of 'region' with your Amazon Cognito Region.
    clientConfig: { region: "us-east-1" },
  }),
});

enum MessageType {
  SUCCESS = 0,
```

```
FAILURE = 1,
EMPTY = 2,
}

const App = () => {
  const [bucketName, setBucketName] = useState("");
  const [msg, setMsg] = useState<{ message: string; type: MessageType }>({
    message: "",
    type: MessageType.EMPTY,
  });

  const createBucket = useCallback(async () => {
    setMsg({ message: "", type: MessageType.EMPTY });

    try {
      await client.send(new CreateBucketCommand({ Bucket: bucketName }));
      setMsg({
        message: `Bucket "${bucketName}" created.`,
        type: MessageType.SUCCESS,
      });
    } catch (e) {
      console.error(e);
      setMsg({
        message: e instanceof Error ? e.message : "Unknown error",
        type: MessageType.FAILURE,
      });
    }
  }, [bucketName]);

  const deleteBucket = useCallback(async () => {
    setMsg({ message: "", type: MessageType.EMPTY });

    try {
      await client.send(new DeleteBucketCommand({ Bucket: bucketName }));
      setMsg({
        message: `Bucket "${bucketName}" deleted.`,
        type: MessageType.SUCCESS,
      });
    } catch (e) {
      setMsg({
        message: e instanceof Error ? e.message : "Unknown error",
        type: MessageType.FAILURE,
      });
    }
  });
}
```

```

    }, [bucketName]);

    return (
      <View style={styles.container}>
        {msg.type !== MessageType.EMPTY && (
          <Text
            style={
              msg.type === MessageType.SUCCESS
                ? styles.successText
                : styles.failureText
            }
          >
            {msg.message}
          </Text>
        )}
        <View>
          <TextInput
            onChangeText={({text}) => setBucketName(text)}
            autoCapitalize={"none"}
            value={bucketName}
            placeholder={"Enter Bucket Name"}
          />
          <Button color="#68a0cf" title="Create Bucket" onPress={createBucket} />
          <Button color="#68a0cf" title="Delete Bucket" onPress={deleteBucket} />
        </View>
      </View>
    );
  };

  const styles = StyleSheet.create({
    container: {
      flex: 1,
      alignItems: "center",
      justifyContent: "center",
    },
    successText: {
      color: "green",
    },
    failureText: {
      color: "red",
    },
  });
});

```

```
export default App;
```

Der Code importiert zunächst die erforderlichen React-, React Native- und AWS SDK-Abhängigkeiten.

In der Funktions-App:

- Das S3Client-Objekt wird erstellt, wobei die Anmeldeinformationen mithilfe des zuvor erstellten Amazon Cognito Identity Pool angegeben werden.
- Die Methoden `createBucket` und `deleteBucket` Erstellen bzw. Löschen des angegebenen Buckets.
- Die React Native-Ansicht zeigt ein Texteingabefeld an, in dem der Benutzer einen Amazon S3 S3-Bucket-Namen angeben kann, sowie Schaltflächen zum Erstellen und Löschen des angegebenen Amazon S3 S3-Buckets.

Die vollständige JavaScript Seite ist [hier verfügbar GitHub](#).

Schritt 6: Führen Sie das Beispiel aus

Note

Denken Sie daran, sich anzumelden! Wenn Sie IAM Identity Center zur Authentifizierung verwenden, denken Sie daran, sich mit dem AWS CLI `aws sso login` folgenden Befehl anzumelden.

Um das Beispiel auszuführen, führen Sie den `android` Befehl `webios`, oder mit `npm` aus.

Hier ist ein Beispiel für die Ausgabe eines `ios` Befehls, der auf macOS ausgeführt wird.

```
$ npm run ios

> ReactNativeApp@0.0.1 ios /Users/trivikr/workspace/ReactNativeApp
> react-native run-ios

info Found Xcode workspace "ReactNativeApp.xcworkspace"
info Launching iPhone 11 (iOS 14.2)
info Building (using "xcodebuild -workspace ReactNativeApp.xcworkspace -configuration
  Debug -scheme ReactNativeApp -destination id=706C1A97-FA38-407D-AD77-CB4FCA9134E9")
```

```
success Successfully built the app
info Installing "/Users/trivikr/Library/Developer/Xcode/DerivedData/ReactNativeApp-
cfhmsyhptwflqqejyspdqgjestr/Build/Products/Debug-iphonesimulator/ReactNativeApp.app"
info Launching "org.reactjs.native.example.ReactNativeApp"

success Successfully launched the app on the simulator
```

Hier ist ein Beispiel für die Ausgabe eines android Befehls, der auf macOS ausgeführt wird.

```
$ npm run android

> ReactNativeApp@0.0.1 android
> react-native run-android

info Running jetifier to migrate libraries to AndroidX. You can disable it using "--no-
jetifier" flag.
Jetifier found 970 file(s) to forward-jetify. Using 12 workers...
info Starting JS server...
info Launching emulator...
info Successfully launched emulator.
info Installing the app...

> Task :app:stripDebugDebugSymbols UP-TO-DATE
Compatible side by side NDK version was not found.

> Task :app:installDebug
02:18:38 V/ddms: execute: running am get-config
02:18:38 V/ddms: execute 'am get-config' on 'emulator-5554' : EOF hit. Read: -1
02:18:38 V/ddms: execute: returning
Installing APK 'app-debug.apk' on 'Pixel_3a_API_30_x86(AVD) - 11' for app:debug
02:18:38 D/app-debug.apk: Uploading app-debug.apk onto device 'emulator-5554'
02:18:38 D/Device: Uploading file onto device 'emulator-5554'
02:18:38 D/ddms: Reading file permission of /Users/trivikr/workspace/ReactNativeApp/
android/app/build/outputs/apk/debug/app-debug.apk as: rw-r--r--
02:18:40 V/ddms: execute: running pm install -r -t "/data/local/tmp/app-debug.apk"
02:18:41 V/ddms: execute 'pm install -r -t "/data/local/tmp/app-debug.apk"' on
'emulator-5554' : EOF hit. Read: -1
02:18:41 V/ddms: execute: returning
02:18:41 V/ddms: execute: running rm "/data/local/tmp/app-debug.apk"
02:18:41 V/ddms: execute 'rm "/data/local/tmp/app-debug.apk"' on 'emulator-5554' : EOF
hit. Read: -1
02:18:41 V/ddms: execute: returning
Installed on 1 device.
```



```
Deprecated Gradle features were used in this build, making it incompatible with Gradle 7.0.
```

```
Use '--warning-mode all' to show the individual deprecation warnings.
```

```
See https://docs.gradle.org/6.2/userguide/command_line_interface.html#sec:command_line_warnings
```

```
BUILD SUCCESSFUL in 6s
```

```
27 actionable tasks: 2 executed, 25 up-to-date
```

```
info Connecting to the development server...
```

```
8081
```

```
info Starting the app on "emulator-5554"...
```

```
Starting: Intent { cmp=com.reactnativeapp/.MainActivity }
```

Geben Sie den Bucket-Namen ein, den Sie erstellen oder löschen möchten, und klicken Sie entweder auf Bucket erstellen oder Bucket löschen. Der entsprechende Befehl wird an Amazon S3 gesendet und eine Erfolgs- oder Fehlermeldung wird angezeigt.

Success: Bucket "test-bucket-name-123" created.

test-bucket-name-123

Create Bucket

Delete Bucket

Mögliche Erweiterungen

Im Folgenden finden Sie Varianten dieser Anwendung, mit denen Sie die Verwendung des AWS SDK JavaScript in einer React Native-App weiter untersuchen können.

- Fügen Sie eine Schaltfläche hinzu, um Amazon S3 S3-Buckets aufzulisten, und stellen Sie neben jedem aufgelisteten Bucket eine Schaltfläche zum Löschen bereit.
- Fügen Sie eine Schaltfläche hinzu, um ein Textobjekt in einen Bucket einzufügen.
- Integrieren Sie einen externen Identitätsanbieter wie Facebook oder Amazon, um ihn mit der authentifizierten IAM-Rolle zu verwenden.

Richten Sie das SDK ein für JavaScript

In den Themen in diesem Abschnitt wird erklärt, wie Sie das SDK installieren und laden, JavaScript damit Sie auf die vom SDK unterstützten Webdienste zugreifen können.

Themen

- [Voraussetzungen](#)
- [Installieren Sie das SDK für JavaScript](#)
- [Laden Sie das SDK für JavaScript](#)

Voraussetzungen

[Installieren Sie Node.js](#). AWS empfiehlt, die Active LTS-Version von Node.js für die Entwicklung zu verwenden.

Themen

- [Richten Sie eine AWS Node.js -Umgebung ein](#)
- [Unterstützte Webbrowser](#)

Richten Sie eine AWS Node.js -Umgebung ein

Verwenden Sie eine der folgenden Methoden, um eine AWS Node.js -Umgebung einzurichten, in der Sie Ihre Anwendung ausführen können:

- Wählen Sie ein Amazon Machine Image (AMI) mit vorinstalliertem Node.js. Erstellen Sie dann eine EC2 Amazon-Instance mit diesem AMI. Wählen Sie bei der Erstellung Ihrer EC2 Amazon-Instance Ihr AMI aus der AWS Marketplace. Suchen Sie nach Node.js und wählen Sie eine AMI-Option, die eine vorinstallierte Version von Node.js (32-Bit oder 64-Bit) enthält. AWS Marketplace
- Erstellen Sie eine EC2 Amazon-Instance und installieren Sie Node.js darauf. Weitere Informationen zur Installation von Node.js auf einer Amazon Linux-Instance finden Sie unter [Node.js auf einer Amazon EC2 EC2-Instance einrichten](#).
- Erstellen Sie eine serverlose Umgebung AWS Lambda , in der Sie Node.js als Lambda-Funktion ausführen. Weitere Informationen zur Verwendung von Node.js in einer Lambda-Funktion finden Sie unter [Programmiermodell \(Node.js\)](#) im AWS Lambda Entwicklerhandbuch.

- Stellen Sie Ihre Node.js -Anwendung auf bereit AWS Elastic Beanstalk. Weitere Informationen zur Verwendung von Node.js mit Elastic Beanstalk finden Sie unter [Deployment Node.js applications to AWS Elastic Beanstalk](#) im AWS Elastic Beanstalk Developer Guide.

Unterstützte Webbrowser

Das AWS SDK für JavaScript unterstützt alle modernen Webbrowser.

In Version 3.567.0 oder höher JavaScript gibt das SDK für ES2 021 Artefakte aus, was die folgenden Mindestversionen unterstützt.

Browser	Version
Google Chrome	85.0+
Mozilla Firefox	80,0 +
Oper	71,0+
Microsoft Edge	85,0+
Apple Safari	14,1+
Samsung Internet	14,0 +

In den Versionen 3.183.0 bis 3.566.0 JavaScript verwendet das SDK für ES2 020 Artefakte, was die folgenden Mindestversionen unterstützt.

Browser	Version
Google Chrome	80,0 und höher
Mozilla Firefox	80,0 +
Oper	63,0+
Microsoft Edge	80,0 +
Apple Safari	14,1+

Browser	Version
Samsung Internet	12,0 +

In Version 3.182.0 oder früher JavaScript verwendet das SDK für ES5 Artefakte, das die folgenden Mindestversionen unterstützt.

Browser	Version
Google Chrome	49,0 +
Mozilla Firefox	45,0 +
Oper	36,0 +
Microsoft Edge	12,0+
Windows Internet Explorer	–
Apple Safari	9,0+
Android-Browser	76,0+
UC-Browser	12,12 +
Samsung Internet	5,0 +

 Note

Frameworks wie bieten AWS Amplify möglicherweise nicht dieselbe Browserunterstützung wie das SDK für JavaScript. Einzelheiten finden Sie in der [AWS Amplify Dokumentation](#).

Installieren Sie das SDK für JavaScript

Nicht alle Dienste sind sofort im SDK oder in allen AWS Regionen verfügbar.

Um einen Dienst AWS SDK für JavaScript mithilfe von [npm, dem Paketmanager Node.js](#), zu installieren, geben Sie an der Befehlszeile den folgenden Befehl *SERVICE* ein, der den Namen eines Dienstes enthält, z. B. s3

```
npm install @aws-sdk/client-SERVICE
```

Eine vollständige Liste der AWS SDK für JavaScript Service-Client-Pakete finden Sie im [AWS SDK für JavaScript API-Referenzhandbuch](#).

Laden Sie das SDK für JavaScript

Nachdem Sie das SDK installiert haben, können Sie mit Hilfe von ein Client-Paket in Ihre Node-Anwendung ladenimport. Um beispielsweise den Amazon S3 S3-Client und den Amazon S3 [ListBuckets](#)S3-Befehl zu laden, verwenden Sie Folgendes.

```
import { S3Client, ListBucketsCommand } from "@aws-sdk/client-s3";
```

Konfigurieren Sie das SDK für JavaScript

Bevor Sie das SDK JavaScript zum Aufrufen von Webdiensten mithilfe der API verwenden, müssen Sie das SDK konfigurieren. Sie müssen mindestens Folgendes konfigurieren:

- Die AWS Region, in der Sie Dienste anfordern werden
- Wie authentifiziert sich Ihr Code mit AWS

Zusätzlich zu diesen Einstellungen müssen Sie möglicherweise auch Berechtigungen für Ihre AWS Ressourcen konfigurieren. Sie können beispielsweise den Zugriff auf einen Amazon S3 S3-Bucket oder den Lesezugriff auf eine Amazon DynamoDB-Tabelle einschränken.

Das [AWS SDKs Referenzhandbuch zu Tools](#) enthält auch Einstellungen, Funktionen und andere grundlegende Konzepte, die vielen von ihnen gemeinsam sind. AWS SDKs

In den Themen dieses Abschnitts wird beschrieben, wie Sie das SDK JavaScript für Node.js konfigurieren und in einem Webbrowser JavaScript ausführen können.

Themen

- [Konfiguration pro Dienst](#)
- [Legen Sie die Region fest AWS](#)
- [Legen Sie die Anmeldeinformationen fest](#)
- [Überlegungen zu Node.js](#)
- [Überlegungen zum Browser-Skript](#)

Konfiguration pro Dienst

Sie können das SDK konfigurieren, indem Sie Konfigurationsinformationen an ein Serviceobjekt übergeben.

Die Konfiguration auf Dienstebene bietet umfassende Kontrolle über einzelne Dienste und ermöglicht es Ihnen, die Konfiguration einzelner Dienstobjekte zu aktualisieren, wenn Ihre Anforderungen von der Standardkonfiguration abweichen.

 Note

In Version 2.x konnte die AWS SDK für JavaScript Dienstkonfiguration an einzelne Client-Konstrukturen übergeben werden. Diese Konfigurationen würden jedoch zunächst automatisch zu einer Kopie der globalen SDK-Konfiguration zusammengeführt. `AWS.config` Außerdem wird `AWS.config.update({/* params */})` nur die aktualisierte Konfiguration für Service-Clients aufgerufen, die nach dem Aktualisierungsauftrag instanziiert wurden, nicht für bestehende Clients.

Dieses Verhalten sorgte häufig für Verwirrung und erschwerte es, dem globalen Objekt eine Konfiguration hinzuzufügen, die sich auf vorwärtskompatible Weise nur auf eine Teilmenge von Service-Clients auswirkt. In Version 3 gibt es keine globale Konfiguration mehr, die vom SDK verwaltet wird. Die Konfiguration muss an jeden Service-Client übergeben werden, der instanziiert wird. Es ist immer noch möglich, dieselbe Konfiguration für mehrere Clients gemeinsam zu verwenden, aber diese Konfiguration wird nicht automatisch mit einem globalen Status zusammengeführt.

Legen Sie die Konfiguration pro Dienst fest

Auf jeden Dienst, den Sie im SDK verwenden, JavaScript wird über ein Dienstobjekt zugegriffen, das Teil der API für diesen Dienst ist. Um beispielsweise auf den Amazon S3 S3-Service zuzugreifen, erstellen Sie das Amazon S3 S3-Serviceobjekt. Sie können für einen bestimmten Service Konfigurationseinstellungen als Teil des Konstruktors für dieses Serviceobjekt definieren.

Wenn Sie beispielsweise auf Amazon EC2-Objekte in mehreren AWS Regionen zugreifen müssen, erstellen Sie ein Amazon EC2-Serviceobjekt für jede Region und legen Sie dann die Regionskonfiguration für jedes Serviceobjekt entsprechend fest.

```
var ec2_regionA = new EC2({region: 'ap-southeast-2', maxAttempts: 15});  
var ec2_regionB = new EC2({region: 'us-west-2', maxAttempts: 15});
```

Legen Sie die Region fest AWS

Eine AWS Region ist eine benannte Gruppe von AWS Ressourcen in demselben geografischen Gebiet. Ein Beispiel für eine Region ist `us-east-1` die Region USA Ost (Nord-Virginia). Sie geben eine Region an, wenn Sie einen Service-Client im SDK für erstellen, JavaScript sodass das SDK

auf den Service in dieser Region zugreift. Einige -Services werden nur in bestimmten Regionen angeboten.

Das SDK für wählt standardmäßig JavaScript keine Region aus. Sie können die AWS Region jedoch mithilfe einer Umgebungsvariablen oder einer gemeinsam genutzten `config` Konfigurationsdatei festlegen.

In einem Client-Klassenkonstruktor

Wenn Sie ein Serviceobjekt instanziiieren, können Sie die AWS Region für diese Ressource als Teil des Client-Klassenkonstruktors angeben, wie hier gezeigt.

```
const s3Client = new S3.S3Client({region: 'us-west-2'});
```

Verwenden Sie eine Umgebungsvariable

Sie können die Region mithilfe der Umgebungsvariablen `AWS_REGION` festlegen. Wenn Sie diese Variable definieren, JavaScript liest das SDK für sie und verwendet sie.

Verwenden Sie eine gemeinsam genutzte Konfigurationsdatei

So wie Sie mit der Datei mit gemeinsam genutzten Anmeldeinformationen Anmeldeinformationen für das SDK speichern können, können Sie Ihre AWS Region und andere Konfigurationseinstellungen in einer gemeinsam genutzten Datei speichern, die `config` nach dem zu verwendenden SDK benannt ist. Wenn die `AWS_SDK_LOAD_CONFIG` Umgebungsvariable auf einen wahrheitsgemäßen Wert gesetzt ist, sucht das SDK für beim Laden JavaScript automatisch nach einer `config` Datei. Wo Sie die `config`-Datei speichern, hängt von Ihrem Betriebssystem ab:

- Linux-, MacOS- oder Unix-Benutzer - `~/.aws/config`
- Windows-Benutzer - `C:\Users\USER_NAME\.aws\config`

Wenn Sie noch keine freigegebene `config`-Datei haben, können Sie diese in dem angegebenen Verzeichnis erstellen. Im folgenden Beispiel werden sowohl die Region als auch das Ausgabeformat über die `config`-Datei definiert.

```
[default]
  region=us-west-2
  output=json
```


Weitere Informationen zur Verwendung von `credentials` Dateien `config` und gemeinsam genutzten Dateien finden Sie unter Dateien [mit gemeinsam genutzten Konfigurationen und Anmeldeinformationen](#) im AWS SDKs Referenzhandbuch für Tools.

Reihenfolge der Festlegung der Region

Im Folgenden finden Sie die Rangfolge für die Einstellung der Region:

1. Wenn eine Region an einen Client-Klassenkonstruktor übergeben wird, wird diese Region verwendet.
2. Wenn in der Umgebungsvariablen eine Region festgelegt ist, wird diese Region verwendet.
3. Andernfalls wird die in der gemeinsam genutzten Konfigurationsdatei definierte Region verwendet.

Legen Sie die Anmeldeinformationen fest

AWS verwendet Anmeldeinformationen, um festzustellen, wer Dienste anruft und ob der Zugriff auf die angeforderten Ressourcen zulässig ist.

Unabhängig davon, ob Ihr JavaScript Code in einem Webbrowser oder auf einem Node.js - Server ausgeführt wird, muss er gültige Anmeldeinformationen erhalten, bevor er über die API auf Dienste zugreifen kann. Anmeldeinformationen können pro Dienst festgelegt werden, indem Anmeldeinformationen direkt an ein Dienstobjekt übergeben werden.

Es gibt mehrere Möglichkeiten, Anmeldeinformationen festzulegen, die sich in Node.js und JavaScript in Webbrowsern unterscheiden. Die Themen in diesem Abschnitt beschreiben, wie Sie die Anmeldeinformationen in Node.js oder Web-Browsern definieren. In jedem Fall werden die Optionen in der empfohlenen Reihenfolge dargestellt.

Bewährte Methoden für Anmeldeinformationen

Durch das ordnungsgemäße Festlegen von Anmeldeinformationen wird sichergestellt, dass Ihre Anwendung oder Ihr Browser-Skript auf die benötigten Services und Ressourcen zugreifen kann. Gleichzeitig werden Sicherheitsrisiken minimiert, die geschäftskritische Anwendungen beeinträchtigen oder vertrauliche Daten kompromittieren könnten.

Ein wichtiger Grundsatz beim Einrichten von Anmeldeinformationen lautet, immer nur die geringstmöglichen Berechtigungen zu erteilen, die für eine Aufgabe erforderlich sind. Es ist sicherer, minimale Berechtigungen für Ihre Ressourcen bereitzustellen und bei Bedarf weitere Berechtigungen hinzuzufügen, anstatt Berechtigungen bereitzustellen, die die geringsten Rechte überschreiten,

und aufgrund dessen zu einem späteren Zeitpunkt möglicherweise Sicherheitsprobleme beheben zu müssen. Wenn Sie beispielsweise nicht einzelne Ressourcen lesen und schreiben müssen, z. B. Objekte in einem Amazon S3 S3-Bucket oder einer DynamoDB-Tabelle, legen Sie für diese Berechtigungen nur Lesen fest.

Weitere Informationen zur Gewährung der geringsten Rechte finden Sie im Abschnitt [Geringste Rechte gewähren](#) des Themas Best Practices im IAM-Benutzerhandbuch.

Themen

- [Legen Sie die Anmeldeinformationen in Node.js fest](#)
- [Anmeldeinformationen in einem Webbrowser einrichten](#)

Legen Sie die Anmeldeinformationen in Node.js fest

Wir empfehlen neuen Benutzern, die sich lokal weiterentwickeln und von ihrem Arbeitgeber keine Authentifizierungsmethode erhalten, zur Einrichtung AWS IAM Identity Center. Weitere Informationen finden Sie unter [SDK-Authentifizierung mit AWS](#).

Es gibt mehrere Möglichkeiten in Node.js, um dem SDK die Anmeldeinformationen bereitzustellen. Einige davon sind sicherer und andere bieten eine höhere Benutzerfreundlichkeit bei der Anwendungsentwicklung. Achten Sie beim Abrufen von Anmeldeinformationen in Node.js darauf, dass Sie sich nicht auf mehr als eine Quelle verlassen, z. B. auf eine Umgebungsvariable und eine JSON-Datei, die Sie laden. Sie können die Berechtigungen, unter denen Ihr Code ausgeführt wird, ändern, ohne sich dieser Änderung bewusst zu sein.

AWS SDK für JavaScript V3 stellt in Node.js eine standardmäßige Anbieterkette für Anmeldeinformationen bereit, sodass Sie nicht explizit einen Anmeldeinformationsanbieter angeben müssen. Die standardmäßige [Anbieterkette für Anmeldeinformationen](#) versucht, die Anmeldeinformationen aus einer Vielzahl verschiedener Quellen in einer bestimmten Rangfolge aufzulösen, bis Anmeldeinformationen aus einer der Quellen zurückgegeben werden. [Die Anmeldeinformationsanbieterkette für SDK für V3 finden Sie hier. JavaScript](#)

Kette der Anbieter von Anmeldeinformationen

Alle SDKs haben eine Reihe von Stellen (oder Quellen), die sie überprüfen, um gültige Anmeldeinformationen zu erhalten, mit denen eine Anfrage an AWS-Service einen gestellt werden kann. Nachdem gültige Anmeldeinformationen gefunden wurden, wird die Suche beendet. Diese systematische Suche wird als Standardanbieterkette für Anmeldeinformationen bezeichnet.

Für jeden Schritt in der Kette gibt es unterschiedliche Möglichkeiten, die Werte festzulegen. Das Setzen von Werten direkt im Code hat immer Vorrang, gefolgt von der Einstellung als Umgebungsvariablen und dann in der gemeinsam genutzten AWS config Datei. Weitere Informationen finden Sie unter [Priorität der Einstellungen](#) im Referenzhandbuch AWS SDKs und im Tools-Referenzhandbuch.

Das Referenzhandbuch AWS SDKs und die Tools enthalten Informationen zu den SDK-Konfigurationseinstellungen, die von allen verwendet werden, AWS SDKs und zu den AWS CLI. Weitere Informationen zur Konfiguration des SDK mithilfe der gemeinsam genutzten AWS config Datei finden Sie unter [Gemeinsam genutzte Konfigurations- und Anmeldeinformationsdateien](#). Weitere Informationen zur Konfiguration des SDK durch das Setzen von Umgebungsvariablen finden Sie unter [Unterstützung von Umgebungsvariablen](#).

Für die Authentifizierung werden AWS die Anbieter der Anmeldeinformationen in der in der folgenden Tabelle angegebenen Reihenfolge AWS SDK für JavaScript überprüft.

AWS SDK für JavaScript API Referenzieren Sie die Methode des Anmeldein- formationsanbieters nach Priorität	Anbieter von Anmeldein- formationen verfügbar	AWS SDKs und Referenzh- andbuch für Tools
fromEnv()	AWS Zugriff auf Schlüssel aus Umgebungsvariablen	AWS Zugriffstasten
fromSSO()	AWS IAM Identity Center. In diesem Handbuch finden Sie weitere Informationen SDK- Authentifizierung mit AWS .	Anbieter von IAM Identity Center-Anmeldeinformationen
fromIni()	AWS Zugriffsschlüssel aus geteilten config Dateien und Dateien credentials	AWS Zugriffstasten
	Vertrauenswürdiger Entitätsa- nbieter (wie <code>AWS_ROLE_ARN</code>)	Nehmen Sie eine IAM-Rolle an

AWS SDK für JavaScript API Referenzieren Sie die Methode des Anmeldein- formationsanbieters nach Priorität	Anbieter von Anmeldein- formationen verfügbar	AWS SDKs und Referenzh- andbuch für Tools
	Web-Identitätstoken von AWS -Security-Token-Service (<code>AWSTokenProvider</code>)	Verbunden mit Web-Identität oder OpenID Connect
	Anmeldeinformationen für Amazon Elastic Container Service (Amazon ECS)	Anbieter von Container- Anmeldeinformationen
	Anmeldeinformationen für das Amazon Elastic Compute Cloud (Amazon EC2) - Instanzprofil (IMDS-Anm- eldeinformationsanbieter)	Anbieter von IMDS-Anme- ldeinformationen
	Anbieter für Prozessan- meldeinformationen	Anbieter für Anmeldein- formationen verarbeiten
	AWS IAM Identity Center	Anbieter von Anmeldein- formationen für das IAM Identity Center
	Anbieter von Anmeldein- formationen	Anbieter von Anmeldein- formationen
fromLoginCredentials()	Anbieter von Anmeldein- formationen	Anbieter von Anmeldein- formationen
fromProcess()	Anbieter für Prozessan- meldeinformationen	Anbieter für Anmeldein- formationen verarbeiten
fromTokenFile()	Web-Identitätstoken von AWS -Security-Token-Service (<code>AWSTokenProvider</code>)	Verbunden mit Web-Identität oder OpenID Connect

AWS SDK für JavaScript API Referenzieren Sie die Methode des Anmeldein- formationsanbieters nach Priorität	Anbieter von Anmeldein- formationen verfügbar	AWS SDKs und Referenzh- andbuch für Tools
fromContainerMetada- ta()	Anmeldeinformationen für Amazon Elastic Container Service (Amazon ECS)	Anbieter von Container- Anmeldeinformationen
fromInstanceMetada- ta()	Anmeldeinformationen für das Amazon Elastic Compute Cloud (Amazon EC2) - Instanzprofil (IMDS-Anm- eldeinformationsanbieter)	Anbieter von IMDS-Anme- Ideinformationen

Wenn Sie für den Einstieg den empfohlenen Ansatz für neue Benutzer befolgt haben, richten Sie die AWS IAM Identity Center Authentifizierung während [SDK-Authentifizierung mit AWS](#) des Themas Erste Schritte ein. Andere Authentifizierungsmethoden sind in verschiedenen Situationen nützlich. Um Sicherheitsrisiken zu vermeiden, empfehlen wir, immer kurzfristige Anmeldeinformationen zu verwenden. Informationen zu anderen Authentifizierungsmethoden finden Sie unter [Authentifizierung und Zugriff](#) im AWS SDKs Referenzhandbuch zu Tools.

Die Themen in diesem Abschnitt beschreiben, wie Sie die Anmeldeinformationen in Node.js laden.

Themen

- [Anmeldeinformationen in Node.js aus IAM-Rollen für Amazon EC2 laden](#)
- [Anmeldeinformationen für eine Lambda-Funktion von Node.js laden](#)

Anmeldeinformationen in Node.js aus IAM-Rollen für Amazon EC2 laden

Wenn Sie Ihre Anwendung Node.js auf einer Amazon EC2-Instance ausführen, können Sie IAM-Rollen für Amazon EC2 nutzen, um automatisch Anmeldeinformationen für die Instance bereitzustellen. Wenn Sie Ihre Instance für die Verwendung von IAM-Rollen konfigurieren, wählt das SDK automatisch die IAM-Anmeldeinformationen für Ihre Anwendung aus, sodass Sie keine Anmeldeinformationen manuell angeben müssen.

Weitere Informationen zum Hinzufügen von IAM-Rollen zu einer Amazon EC2-Instance finden Sie unter [IAM-Rollen für Amazon EC2](#).

Anmeldeinformationen für eine Lambda-Funktion von Node.js laden

Wenn Sie eine AWS Lambda Funktion erstellen, müssen Sie eine spezielle IAM-Rolle erstellen, die über die Berechtigung zum Ausführen der Funktion verfügt. Diese Rolle wird Ausführungsrolle genannt. Wenn Sie eine Lambda-Funktion einrichten, müssen Sie die von Ihnen erstellte IAM-Rolle als entsprechende Ausführungsrolle angeben.

Die Ausführungsrolle stellt der Lambda-Funktion die Anmeldeinformationen zur Verfügung, die sie zum Ausführen und Aufrufen anderer Webdienste benötigt. Daher müssen Sie keine Anmeldeinformationen für den Code Node.js angeben, den Sie in einer Lambda-Funktion schreiben.

Weitere Informationen zum Erstellen einer Lambda-Ausführungsrolle finden Sie unter [Berechtigungen verwalten: Verwenden einer IAM-Rolle \(Ausführungsrolle\)](#) im AWS Lambda Entwicklerhandbuch.

Anmeldeinformationen in einem Webbrowser einrichten

Es gibt mehrere Möglichkeiten, um dem SDK die Anmeldeinformationen aus Browser-Skripts bereitzustellen. Einige davon sind sicherer und andere bieten eine höhere Benutzerfreundlichkeit bei der Entwicklung eines Skripts.

So können Sie Ihre Anmeldeinformationen in der Reihenfolge ihrer Empfehlungen angeben:

1. Verwenden von Amazon Cognito Identity zur Benutzerauthentifizierung und Bereitstellung von Anmeldeinformationen
2. Verwenden von Web-Verbundidentitäten

Warning

Wir empfehlen nicht, Ihre AWS Anmeldeinformationen in Ihren Skripten fest zu codieren. Die Hartcodierung der Anmeldeinformationen setzt die Zugriffsschlüssel-ID und den geheimen Zugriffsschlüssel dem Risiko einer Offenlegung aus.

Themen

- [Verwenden Sie Amazon Cognito Identity, um Benutzer zu authentifizieren](#)

Verwenden Sie Amazon Cognito Identity, um Benutzer zu authentifizieren

Die empfohlene Methode zum Abrufen von AWS Anmeldeinformationen für Ihre Browserskripte ist die Verwendung des Amazon Cognito Identity-Anmeldeinformationsclients `CognitoIdentityClient`. Amazon Cognito ermöglicht die Authentifizierung von Benutzern über externe Identitätsanbieter.

Um Amazon Cognito Identity verwenden zu können, müssen Sie zunächst einen Identitätspool in der Amazon Cognito Cognito-Konsole erstellen. Ein Identitäten-Pool stellt die Gruppe von Identitäten dar, die Ihre Anwendung für Ihre Benutzer bereitstellt. Die den Benutzern zugewiesenen Identitäten identifizieren jedes Benutzerkonto eindeutig. Amazon-Cognito-Identitäten sind keine Anmeldeinformationen. Sie werden mithilfe der Unterstützung von Web Identity Federation in AWS - Security-Token-Service (AWS STS) gegen Anmeldeinformationen ausgetauscht.

Amazon Cognito unterstützt Sie bei der Verwaltung der Abstraktion von Identitäten über mehrere Identitätsanbieter hinweg. Die geladene Identität wird dann gegen Anmeldeinformationen in AWS STS ausgetauscht.

Konfigurieren Sie das Amazon Cognito Identity-Anmeldeinformationsobjekt

Wenn Sie noch keinen erstellt haben, erstellen Sie einen Identitätspool, den Sie mit Ihren Browserskripten in der [Amazon Cognito Cognito-Konsole](#) verwenden können, bevor Sie Ihren Amazon Cognito Cognito-Client konfigurieren. Erstellen Sie sowohl authentifizierte als auch nicht authentifizierte IAM-Rollen für Ihren Identitätspool und ordnen Sie sie zu. Weitere Informationen finden Sie unter [Tutorial: Einen Identitätspool erstellen](#) im Amazon Cognito Developer Guide.

Bei nicht authentifizierten Benutzern wird die Identität nicht verifiziert, sodass diese Rolle für Gastbenutzer Ihrer App oder für Fälle geeignet ist, in denen es egal ist, ob die Identität der Benutzer verifiziert wurde. Authentifizierte Benutzer melden sich bei Ihrer Anwendung über einen Drittanbieter an, der ihre Identität überprüft. Vergewissern Sie sich, dass Sie die Berechtigungen der Ressourcen entsprechend anpassen, damit Sie keinen Zugriff von nicht authentifizierten Benutzern darauf gewähren.

Nachdem Sie einen Identitätspool konfiguriert haben, verwenden Sie die `fromCognitoIdentityPool` Methode von, `@aws-sdk/credential-providers` um die Anmeldeinformationen aus dem Identitätspool abzurufen. Ersetzen Sie im folgenden Beispiel für die Erstellung eines Amazon S3 S3-Clients `AWS_REGION` durch die Region und `IDENTITY_POOL_ID` die Identitätspool-ID.

```
// Import required AWS SDK clients and command for Node.js
import {S3Client} from "@aws-sdk/client-s3";
import {fromCognitoIdentityPool} from "@aws-sdk/credential-providers";

const REGION = AWS_REGION;

const s3Client = new S3Client({
  region: REGION,
  credentials: fromCognitoIdentityPool({
    clientConfig: { region: REGION }, // Configure the underlying
    CognitoIdentityClient.
    identityPoolId: 'IDENTITY_POOL_ID',
    logins: {
      // Optional tokens, used for authenticated login.
    },
  })
});
```

Die optionale `logins`-Eigenschaft ist eine Abbildung der Namen des Identitätsanbieters auf die Identitäts-Token für diese Anbieter. Wie Sie den Token von Ihrem Identitätsanbieter erhalten, hängt davon ab, welchen Anbieter Sie verwenden. Wenn Sie beispielsweise einen Amazon Cognito Cognito-Benutzerpool als Authentifizierungsanbieter verwenden, könnten Sie eine Methode verwenden, die der folgenden ähnelt.

```
// Get the Amazon Cognito ID token for the user. 'getToken()' below.
let idToken = getToken();
let COGNITO_ID = "COGNITO_ID"; // 'COGNITO_ID' has the format 'cognito-
idp.REGION.amazonaws.com/COGNITO_USER_POOL_ID'
let loginData = {
  [COGNITO_ID]: idToken,
};
const s3Client = new S3Client({
  region: REGION,
  credentials: fromCognitoIdentityPool({
    clientConfig: { region: REGION }, // Configure the underlying
    CognitoIdentityClient.
    identityPoolId: 'IDENTITY_POOL_ID',
    logins: loginData
  })
});

// Strips the token ID from the URL after authentication.
window.getToken = function () {
```



```
var idtoken = window.location.href;
var idtoken1 = idtoken.split("=")[1];
var idtoken2 = idtoken1.split("&")[0];
var idtoken3 = idtoken2.split("&")[0];
return idtoken3;
};
```

Wechseln Sie zwischen nicht authentifizierten Benutzern und authentifizierten Benutzern

Amazon Cognito unterstützt sowohl authentifizierte als auch nicht authentifizierte Benutzer. Nicht authentifizierte Benutzer erhalten Zugriff auf Ihre Ressourcen, auch wenn sie nicht über Ihre Identitätsanbieter angemeldet sind. Dieser Grad des Zugriffs ist nützlich, um Inhalte für Benutzer anzuzeigen, bevor diese sich anmelden. Jeder nicht authentifizierte Benutzer hat eine eindeutige Identität in Amazon Cognito, obwohl er nicht einzeln angemeldet und authentifiziert wurde.

Anfänglich nicht authentifizierter Benutzer

Benutzer beginnen in der Regel mit der nicht authentifizierten Rolle, für die Sie die Eigenschaft für die Anmeldeinformationen Ihres Konfigurationsobjekts ohne eine `logins`-Eigenschaft festlegen. In diesem Fall könnten Ihre Standardanmeldedaten wie folgt aussehen:

```
// Import the required AWS SDK für JavaScript v3 modules.
import {fromCognitoIdentityPool} from "@aws-sdk/credential-providers";
// Set the default credentials.
const creds = fromCognitoIdentityPool({
  identityPoolId: 'IDENTITY_POOL_ID',
  clientConfig: { region: REGION } // Configure the underlying CognitoIdentityClient.
});
```

Wechseln Sie zum authentifizierten Benutzer

Wenn sich ein nicht authentifizierter Benutzer bei einem Identitätsanbieter anmeldet und Sie über ein Token verfügen, können Sie den Benutzer von nicht authentifiziert auf authentifiziert umstellen, indem Sie eine benutzerdefinierte Funktion aufrufen, die das Anmeldeinformationsobjekt aktualisiert und das Token hinzufügt. `logins`

```
// Called when an identity provider has a token for a logged in user
function userLoggedIn(providerName, token) {
  creds.params.Logins = creds.params.logins || {};
  creds.params.Logins[providerName] = token;
```

```
// Expire credentials to refresh them on the next request
creds.expired = true;
}
```

Überlegungen zu Node.js

Der Code Node.js ist es zwar JavaScript, aber die Verwendung von AWS SDK für JavaScript in Node.js kann sich von der Verwendung des SDK in Browserskripts unterscheiden. Einige API-Methoden funktionieren in Node.js, jedoch nicht in Browser-Skripts und umgekehrt. Und die erfolgreiche Verwendung einiger Module APIs hängt davon ab, ob Sie mit den gängigen Node.js - Codierungsmustern vertraut sind, z. B. mit dem Import und der Verwendung anderer Node.js -Module wie dem File System (fs) Modul.

Note

AWS empfiehlt die Verwendung der Active LTS-Version von Node.js für die Entwicklung.

Verwenden Sie die integrierten Node.js -Module

Node.js bietet eine Reihe von integrierten Modulen, die Sie verwenden können ohne sie installieren zu müssen. Um diese Module verwenden zu können, müssen Sie ein Objekt mit der `require`-Methode erstellen und den Modulnamen angeben. Wenn beispielsweise das integrierte HTTP-Modul enthalten sein soll, geben Sie Folgendes ein.

```
import http from 'http';
```

Rufen Sie Methoden des Moduls ab, als würde es sich um Methoden dieses Objekts handeln. Das folgende Beispiel zeigt Code, der eine HTML-Datei liest.

```
// include File System module
import fs from "fs";
// Invoke readFile method
fs.readFile('index.html', function(err, data) {
  if (err) {
    throw err;
  } else {
    // Successful file read
  }
}
```

```
}  
});
```

Eine vollständige Liste aller integrierten Module, die Node.js bereitstellt, finden Sie in der [Node.js - Dokumentation](#) auf der Website Node.js.

Verwenden Sie npm-Pakete

Zusätzlich zu den integrierten Modulen können Sie auch Code von Drittanbietern aus dem npm Paketmanager Node.js einbinden und integrieren. Hierbei handelt es sich um ein Repository mit Open-Source-Node.js-Paketen sowie um eine Befehlszeilen-Schnittstelle für die Installation dieser Pakete. Weitere Informationen zu npm und eine Liste der derzeit verfügbaren Pakete finden Sie unter <https://www.npmjs.com>. Weitere Informationen zu den zusätzlichen Paketen von Node.js, die Sie verwenden können, finden Sie [hier auf GitHub](#).

MaxSockets in Node.js konfigurieren

In Node.js können Sie die maximale Anzahl der Verbindungen pro Ursprungsserver festlegen. Wenn `maxSockets` festgelegt wurde, setzt der Low-Level-HTTP-Client die Anforderungen auf eine Warteliste und ordnet sie Sockets zu, sobald diese verfügbar werden.

Mit dieser Option können Sie eine Obergrenze für die Anzahl der gleichzeitigen Anforderungen an einen bestimmten Ursprungsserver angeben. Indem Sie diesen Wert senken, kann die Anzahl der maximalen Ablehnungen oder Timeout-Fehler reduziert werden. Es kann jedoch auch die Speichernutzung erhöhen, da Anforderungen so lange in eine Warteschlange gesetzt werden, bis ein Socket verfügbar wird.

Das folgende Beispiel zeigt, wie die Einstellungen `maxSockets` für einen DynamoDB-Client vorgenommen werden.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";  
import { NodeHttpHandler } from "@smithy/node-http-handler";  
import https from "https";  
let agent = new https.Agent({  
  maxSockets: 25  
});  
  
let dynamodbClient = new DynamoDBClient({  
  requestHandler: new NodeHttpHandler({  
    requestTimeout: 3_000,  
    httpsAgent: agent  
  })  
});
```

```
});  
});
```

Das SDK für JavaScript verwendet den `maxSockets` Wert 50, wenn Sie keinen Wert oder kein Objekt `Agent` angeben. Wenn Sie ein `Agent` Objekt angeben, wird sein `maxSockets` Wert verwendet. Weitere Informationen zur Einstellung `maxSockets` in Node.js finden Sie in der [Dokumentation zu Node.js](#).

Ab Version 3.521.0 von können Sie AWS SDK für JavaScript die folgende [Kurzsyntax](#) zur Konfiguration verwenden. `requestHandler`

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";  
  
const client = new DynamoDBClient({  
  requestHandler: {  
    requestTimeout: 3_000,  
    httpsAgent: { maxSockets: 25 },  
  },  
});
```

Verbindungen mit Keep-Alive in Node.js wiederverwenden

Der HTTP/HTTPS Standardagent Node.js erstellt für jede neue Anfrage eine neue TCP-Verbindung. Um die Kosten für den Aufbau einer neuen Verbindung zu vermeiden, werden TCP-Verbindungen standardmäßig AWS SDK für JavaScript wiederverwendet.

Bei kurzlebigen Vorgängen, wie Amazon DynamoDB DynamoDB-Abfragen, kann der Latenzaufwand beim Einrichten einer TCP-Verbindung größer sein als der Vorgang selbst. Da die [DynamoDB-Verschlüsselung im Ruhezustand](#) integriert ist [AWS KMS](#), kann es außerdem zu Latenzen kommen, weil die Datenbank für jeden Vorgang neue AWS KMS Cacheeinträge einrichten muss.

Wenn Sie TCP-Verbindungen nicht wiederverwenden möchten, können Sie die Wiederverwendung dieser Verbindungen `keepAlive` live with für jeden einzelnen Service deaktivieren, wie im folgenden Beispiel für einen DynamoDB-Client gezeigt.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";  
import { NodeHttpHandler } from "@smithy/node-http-handler";  
import { Agent } from "https";  
  
const dynamodbClient = new DynamoDBClient({  
  requestHandler: new NodeHttpHandler({
```

```
    httpsAgent: new Agent({ keepAlive: false })
  })
});
```

Wenn `keepAlive` aktiviert, können Sie auch die Anfangsverzögerung für TCP-Keep-Alive-Pakete mit `keepAliveMsecs` festlegen, die standardmäßig 1000 ms beträgt. Weitere Informationen finden Sie in der [Node.js-Dokumentation](#).

Konfigurieren Sie Proxys für Node.js

Wenn Sie keine direkte Verbindung zum Internet herstellen können, unterstützt das SDK für die Verwendung von HTTP- oder HTTPS-Proxys über einen HTTP-Agenten eines Drittanbieters.

[Um einen HTTP-Agenten eines Drittanbieters zu finden, suchen Sie bei npm nach „HTTP-Proxy“.](#)

Um einen HTTP-Agent-Proxy eines Drittanbieters zu installieren, geben Sie in der Befehlszeile Folgendes ein, wobei der Name des npm Pakets angegeben *PROXY* ist.

```
npm install PROXY --save
```

Um einen Proxy in Ihrer Anwendung zu verwenden, verwenden Sie die `httpsAgent` Eigenschaft `httpAgent` and, wie im folgenden Beispiel für einen DynamoDB-Client gezeigt.

```
import { DynamoDBClient } from '@aws-sdk/client-dynamodb';
import { NodeHttpHandler } from '@smithy/node-http-handler';
import { HttpsProxyAgent } from 'hpagent';
const agent = new HttpsProxyAgent({ proxy: "http://internal.proxy.com" });
const dynamodbClient = new DynamoDBClient({
  requestHandler: new NodeHttpHandler({
    httpAgent: agent,
    httpsAgent: agent
  }),
});
```

Note

`httpAgent` ist nicht dasselbe wie `httpsAgent`, und da die meisten Aufrufe vom Client an `https` erfolgen, sollten beide gesetzt werden.

Registrieren Sie Zertifikatspakete in Node.js

Die Standard-Vertrauensspeicher für Node.js enthalten die Zertifikate, die für den Zugriff auf AWS Dienste erforderlich sind. In einigen Fällen könnte es vorteilhaft sein, nur eine bestimmte Gruppe von Zertifikaten einzuschließen.

In diesem Beispiel wird ein bestimmtes Zertifikat auf der Festplatte verwendet, mit dem ein `https.Agent` erstellt wird, der alle Verbindungen ablehnt, die nicht über das vorgesehene Zertifikat verfügen. Das neu erstellte Objekt `https.Agent` wird dann vom DynamoDB-Client verwendet.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { NodeHttpHandler } from "@smithy/node-http-handler";
import { Agent } from "https";
import { readFileSync } from "fs";
const certs = [readFileSync("/path/to/cert.pem")];
const agent = new Agent({
  rejectUnauthorized: true,
  ca: certs
});
const dynamoDbClient = new DynamoDBClient({
  requestHandler: new NodeHttpHandler({
    httpAgent: agent,
    httpsAgent: agent
  })
});
```

Überlegungen zum Browser-Skript

In den folgenden Themen werden besondere Überlegungen zur Verwendung von AWS SDK für JavaScript In-Browser-Skripts beschrieben.

Themen

- [Erstellen Sie das SDK für Browser](#)
- [Cross-Origin Resource Sharing \(CORS\)](#)
- [Bündeln Sie Anwendungen mit Webpack](#)

Erstellen Sie das SDK für Browser

Im Gegensatz zum SDK für JavaScript Version 2 (V2) wird V3 nicht als JavaScript Datei mit Unterstützung für eine Reihe von Standarddiensten bereitgestellt. Stattdessen können Sie mit V3 nur das SDK für die benötigten JavaScript Dateien bündeln und in den Browser aufnehmen, wodurch der Overhead reduziert wird. Wir empfehlen, Webpack zu verwenden, um das erforderliche SDK für JavaScript Dateien und alle zusätzlichen Pakete von Drittanbietern, die Sie benötigen, in einer einzigen Javascript Datei zu bündeln und diese mithilfe eines `<script>` Tags in Browserskripte zu laden. Weitere Informationen zu Webpack finden Sie unter [Bündeln Sie Anwendungen mit Webpack](#)

Wenn Sie mit dem SDK außerhalb einer Umgebung arbeiten, die CORS in Ihrem Browser erzwingt, und wenn Sie Zugriff auf alle vom SDK bereitgestellten Dienste für haben möchten JavaScript, können Sie lokal eine benutzerdefinierte Kopie des SDK erstellen, indem Sie das Repository klonen und dieselben Build-Tools ausführen, mit denen die gehostete Standardversion des SDK erstellt wird. In den folgenden Abschnitten werden die Schritte erklärt, anhand derer das SDK zusammen mit zusätzlichen Services und API-Versionen erstellt werden kann.

Verwenden Sie den SDK Builder, um das SDK für zu erstellen JavaScript

Note

Amazon Web Services Version 3 (V3) unterstützt Browser Builder nicht mehr. Um die Bandbreitennutzung von Browseranwendungen zu minimieren, empfehlen wir Ihnen, benannte Module zu importieren und sie zu bündeln, um die Größe zu reduzieren. Weitere Informationen zur Bündelung finden Sie unter [Bündeln Sie Anwendungen mit Webpack](#).

Cross-Origin Resource Sharing (CORS)

Bei Cross-Origin Resource Sharing oder CORS handelt es sich um eine Sicherheitsfunktion der modernen Webbrowser. Es ermöglicht Webbrowsern zu verhandeln, welche Domänen Anforderungen von externen Websites oder Services senden können.

CORS ist ein wichtiger Aspekt bei der Entwicklung von Browseranwendungen mit dem AWS SDK für JavaScript, da die meisten Anforderungen an Ressourcen an eine externe Domäne, wie z. B. der Endpunkt für einen Webservice, gesendet werden. Wenn in Ihrer JavaScript Umgebung die CORS-Sicherheit durchgesetzt wird, müssen Sie CORS mit dem Dienst konfigurieren.

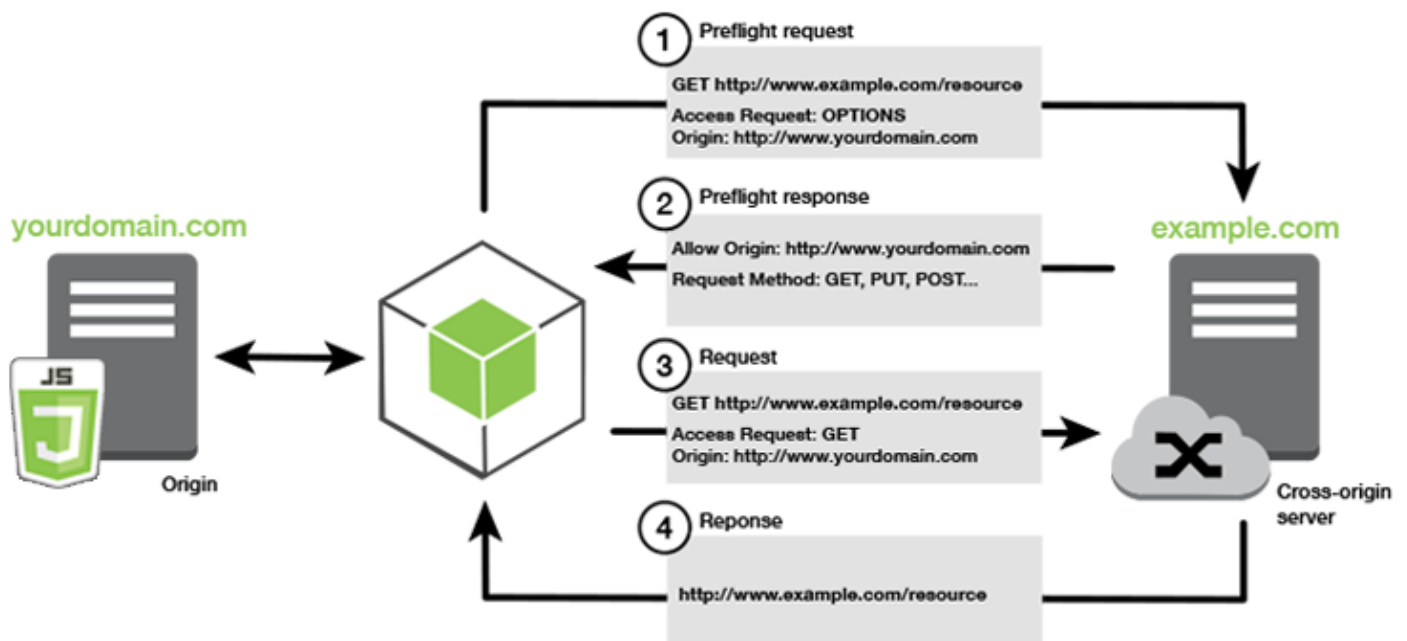
CORS bestimmt auf der Grundlage der folgenden Kriterien, ob die gemeinsame Nutzung von Ressourcen in einer ursprungsübergreifenden Anfrage zulässig ist:

- Der spezifischen Domäne, die die Anforderung sendet
- Dem Typ der HTTP-Anforderung, die gesendet wird (GET, PUT, POST, DELETE usw.)

Wie funktioniert CORS

Im einfachsten Fall sendet Ihr Browser-Skript eine GET-Anforderung für eine Ressource von einem Server in einer anderen Domäne. Abhängig von der CORS-Konfiguration dieses Servers, wenn die Anforderung von einer Domäne stammt, die berechtigt ist, GET-Anforderungen abzusenden, reagiert der ursprungsübergreifende Server, indem er die angeforderte Ressource zurück gibt.

Wenn entweder die anfordernde Domäne oder der Typ der HTTP-Anforderung nicht autorisiert ist, wird die Anforderung abgelehnt. CORS jedoch ermöglicht ein Preflight der Anforderung, bevor sie tatsächlich abgesendet wird. In diesem Fall wird eine Preflight-Anforderung ausgeführt, in der die OPTIONS-Zugriffsanforderungsoperation gesendet wird. Wenn die CORS-Konfiguration des ursprungsübergreifenden Servers Zugriff auf die anfordernde Domäne gewährt, sendet der Server eine Preflight-Antwort zurück, die alle HTTP-Anforderungstypen aufführt, die die anfordernde Domäne für die angeforderte Ressource vornehmen kann.



Ist eine CORS-Konfiguration erforderlich?

Amazon S3 S3-Buckets benötigen eine CORS-Konfiguration, bevor Sie Operationen mit ihnen ausführen können. In einigen JavaScript Umgebungen wird CORS möglicherweise nicht durchgesetzt, sodass die Konfiguration von CORS nicht erforderlich ist. Wenn Sie Ihre Anwendung beispielsweise von einem Amazon S3 S3-Bucket aus hosten und auf Ressourcen von `*.s3.amazonaws.com` oder einem anderen bestimmten Endpunkt zugreifen, greifen Ihre Anfragen nicht auf eine externe Domain zu. Aus diesem Grund benötigt diese Konfiguration kein CORS. In diesem Fall wird CORS weiterhin für andere Dienste als Amazon S3 verwendet.

CORS für einen Amazon S3 S3-Bucket konfigurieren

Sie können einen Amazon S3 S3-Bucket für die Verwendung von CORS in der Amazon S3 S3-Konsole konfigurieren.

Wenn Sie CORS in der AWS Web Services Management Console konfigurieren, müssen Sie JSON verwenden, um eine CORS-Konfiguration zu erstellen. Die neue AWS Web Services Management Console unterstützt nur JSON-CORS-Konfigurationen.

Important

In der neuen AWS Web Services Management Console muss die CORS-Konfiguration JSON sein.

1. Öffnen Sie in der AWS Web Services Management Console die Amazon S3 S3-Konsole, suchen Sie den Bucket, den Sie konfigurieren möchten, und aktivieren Sie das entsprechende Kontrollkästchen.
2. Wählen Sie in dem sich öffnenden Bereich Permissions aus.
3. Wählen Sie auf der Registerkarte Permission die Option CORS-Konfiguration aus.
4. Geben Sie Ihre CORS-Konfiguration im CORS-Konfigurationseditor ein und wählen Sie dann Speichern.

Eine CORS-Konfiguration ist eine XML-Datei mit einer Reihe von Regeln innerhalb eines `<CORSRule>`. Eine Konfiguration kann bis zu 100 Regeln enthalten. Eine Regel wird durch eines der folgenden Tags definiert:

- **<AllowedOrigin>**— Gibt die Domain-Ursprünge an, die Sie für domänenübergreifende Anfragen zulassen.
- **<AllowedMethod>**— Gibt einen Anfragetyp an, den Sie in domänenübergreifenden Anfragen zulassen (GET, PUT, POST, DELETE, HEAD).
- **<AllowedHeader>**— Gibt die Header an, die in einer Preflight-Anfrage zulässig sind.

Beispielkonfigurationen finden Sie unter [Wie konfiguriere ich CORS in meinem Bucket?](#) im Amazon Simple Storage Service-Benutzerhandbuch.

Beispiel für eine CORS-Konfiguration

Das folgende CORS-Konfigurationsbeispiel ermöglicht es einem Benutzer, Objekte innerhalb eines Buckets aus der Domain anzuzeigen, hinzuzufügen, zu entfernen oder zu aktualisieren. `example.org` Wir empfehlen jedoch, dass Sie den Bereich **<AllowedOrigin>** auf die Domain Ihrer Website beschränken. Sie können "*" angeben, sodass jeder Ursprung zulässig ist.

Important

In der neuen S3-Konsole muss die CORS-Konfiguration JSON sein.

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<CORSConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <CORSRule>
    <AllowedOrigin>https://example.org</AllowedOrigin>
    <AllowedMethod>HEAD</AllowedMethod>
    <AllowedMethod>GET</AllowedMethod>
    <AllowedMethod>PUT</AllowedMethod>
    <AllowedMethod>POST</AllowedMethod>
    <AllowedMethod>DELETE</AllowedMethod>
    <AllowedHeader>*</AllowedHeader>
    <ExposeHeader>ETag</ExposeHeader>
    <ExposeHeader>x-amz-meta-custom-header</ExposeHeader>
  </CORSRule>
</CORSConfiguration>
```

JSON

```
[
  {
    "AllowedHeaders": [
      "*"
    ],
    "AllowedMethods": [
      "HEAD",
      "GET",
      "PUT",
      "POST",
      "DELETE"
    ],
    "AllowedOrigins": [
      "https://www.example.org"
    ],
    "ExposeHeaders": [
      "ETag",
      "x-amz-meta-custom-header"
    ]
  }
]
```

Diese Konfiguration autorisiert den Benutzer nicht, Aktionen für den Bucket auszuführen. Es ermöglicht dem Sicherheitsmodell des Browsers, eine Anfrage an Amazon S3 zuzulassen. Berechtigungen müssen über Bucket-Berechtigungen oder IAM-Rollenberechtigungen konfiguriert werden.

Sie können `ExposeHeader` damit das SDK die von Amazon S3 zurückgegebenen Antwort-Header lesen lassen. Wenn Sie beispielsweise den `ETag` Header aus einem `PUT` oder einem mehrteiligen Upload lesen möchten, müssen Sie das `ExposeHeader` Tag in Ihre Konfiguration aufnehmen, wie im vorherigen Beispiel gezeigt. Das SDK kann nur auf Header zugreifen, die über die CORS-Konfiguration bereitgestellt werden. Wenn Sie Metadaten für das Objekt festlegen, werden Werte als Header mit dem Präfix `x-amz-meta-` zurückgegeben, wie z. B. `x-amz-meta-my-custom-header`, und müssen auch auf die gleiche Weise bereitgestellt werden.

Bündeln Sie Anwendungen mit Webpack

Die Verwendung von Codemodulen durch Webanwendungen in Browserskripten oder Node.js erzeugt Abhängigkeiten. Diese Codemodule können eigene Abhängigkeiten haben, was zu einer

Sammlung von miteinander verbundenen Modulen führt, die Ihre Anwendung benötigt, um zu funktionieren. Um Abhängigkeiten zu verwalten, können Sie einen Modul-Bundler wie webpack verwenden.

Der webpack Modul-Bundler analysiert Ihren Anwendungscode und sucht nach `import` `require` ODER-Anweisungen, um Bundles zu erstellen, die alle Ressourcen enthalten, die Ihre Anwendung benötigt. Auf diese Weise können die Ressourcen einfach über eine Webseite bereitgestellt werden. Das SDK für JavaScript kann webpack als eine der Abhängigkeiten in das Ausgabepaket aufgenommen werden.

Weitere Informationen zu webpack finden Sie im [Webpack-Modul-Bundler](#) unter. GitHub

Installieren Sie das Webpack

Um den webpack Modul-Bundler zu installieren, müssen Sie zuerst npm, den Paketmanager Node.js, installiert haben. Geben Sie den folgenden Befehl ein, um die webpack CLI und das JavaScript Modul zu installieren.

```
npm install --save-dev webpack
```

Um das path Modul für die Arbeit mit Datei- und Verzeichnispfaden zu verwenden, das automatisch mit Webpack installiert wird, müssen Sie möglicherweise das path-browserify Paket Node.js installieren.

```
npm install --save-dev path-browserify
```

Webpack konfigurieren

Standardmäßig sucht Webpack nach einer JavaScript Datei mit dem Namen `webpack.config.js` im Stammverzeichnis Ihres Projekts. Diese Datei enthält Ihre Konfigurationsoptionen. Das Folgende ist ein Beispiel für eine `webpack.config.js` Konfigurationsdatei für WebPack Version 5.0.0 und höher.

Note

Die Webpack-Konfigurationsanforderungen variieren je nach der Version von Webpack, die Sie installieren. Weitere Informationen finden Sie in der [Webpack-Dokumentation](#).

```
// Import path for resolving file paths
var path = require("path");
module.exports = {
  // Specify the entry point for our app.
  entry: [path.join(__dirname, "browser.js")],
  // Specify the output file containing our bundled code.
  output: {
    path: __dirname,
    filename: 'bundle.js'
  },
  // Enable WebPack to use the 'path' package.
  resolve:{
  fallback: { path: require.resolve("path-browserify")}
  }
  /**
   * In Webpack version v2.0.0 and earlier, you must tell
   * webpack how to use "json-loader" to load 'json' files.
   * To do this Enter 'npm --save-dev install json-loader' at the
   * command line to install the "json-loader" package, and include the
   * following entry in your webpack.config.js.
   * module: {
   *   rules: [{test: /\.json$/, use: use: "json-loader"}]
   * }
   */
};
```

In diesem Beispiel `browser.js` wird als Einstiegspunkt angegeben. Der Einstiegspunkt wird von der Datei `webpack` verwendet, um mit der Suche nach importierten Modulen zu beginnen. Der Name für die Ausgabedatei ist als `bundle.js` angegeben. Diese Ausgabedatei enthält alles, was JavaScript die Anwendung zum Ausführen benötigt. Wenn der im Einstiegspunkt angegebene Code andere Module importiert oder benötigt, z. B. das SDK für JavaScript, wird dieser Code gebündelt, ohne dass er in der Konfiguration angegeben werden muss.

Führen Sie Webpack aus

Um eine zu verwendende Anwendung zu erstellen `webpack`, fügen Sie dem `scripts` Objekt in Ihrer `package.json` Datei Folgendes hinzu.

```
"build": "webpack"
```

Im Folgenden finden Sie eine `package.json` Beispieldatei, die das Hinzufügen demonstriert `webpack`.

```
{
  "name": "aws-webpack",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "build": "webpack"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "@aws-sdk/client-iam": "^3.32.0",
    "@aws-sdk/client-s3": "^3.32.0"
  },
  "devDependencies": {
    "webpack": "^5.0.0"
  }
}
```

Geben Sie den folgenden Befehl ein, um Ihre Anwendung zu erstellen.

```
npm run build
```

Der `webpack` Modul-Bundler generiert dann die JavaScript Datei, die Sie im Stammverzeichnis Ihres Projekts angegeben haben.

Verwenden Sie das Webpack-Bundle

Um das Bundle in einem Browser-Skript zu verwenden, können Sie das Bundle mithilfe eines `<script>` Tags integrieren, wie im folgenden Beispiel gezeigt.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Amazon SDK with webpack</title>
  </head>
  <body>
    <div id="list"></div>
```

```
    <script src="bundle.js"></script>
  </body>
</html>
```

Paket für Node.js

Sie können webpack es verwenden, um Bundles zu generieren, die in Node.js ausgeführt werden, indem Sie es node als Ziel in der Konfiguration angeben.

```
target: "node"
```

Dies ist nützlich, wenn eine Node.js-Anwendung in einer Umgebung ausgeführt wird, in der Speicherplatz begrenzt ist. Hier sehen Sie ein Beispiel für eine `webpack.config.js`-Konfiguration mit der Angabe von Node.js als Ausgabeziel.

```
// Import path for resolving file paths
var path = require("path");
module.exports = {
  // Specify the entry point for our app.
  entry: [path.join(__dirname, "browser.js")],
  // Specify the output file containing our bundled code.
  output: {
    path: __dirname,
    filename: 'bundle.js'
  },
  // Let webpack know to generate a Node.js bundle.
  target: "node",
  // Enable WebPack to use the 'path' package.
  resolve: {
    fallback: { path: require.resolve("path-browserify")}
  },
  /**
   * In Webpack version v2.0.0 and earlier, you must tell
   * webpack how to use "json-loader" to load 'json' files.
   * To do this Enter 'npm --save-dev install json-loader' at the
   * command line to install the "json-loader" package, and include the
   * following entry in your webpack.config.js.
   */
  module: {
    rules: [{test: /\.json$/, use: use: "json-loader"}]
  }
};
```

Arbeiten Sie mit AWS Diensten im SDK für JavaScript

Die AWS SDK für JavaScript Version 3 bietet Zugriff auf Dienste, die sie über eine Sammlung von Client-Klassen unterstützt. Von diesen Client-Klassen erstellen Sie Service-Schnittstellenobjekte, die allgemein als Service-Objekte bezeichnet werden. Jeder unterstützte AWS Dienst verfügt über eine oder mehrere Clientklassen, die eine einfache Nutzung APIs von Servicefunktionen und Ressourcen ermöglichen. Amazon DynamoDB ist beispielsweise über APIs die DynamoDB Klasse verfügbar.

Die über das SDK bereitgestellten Dienste JavaScript folgen dem Anfrage-Antwort-Muster, um Nachrichten mit aufrufenden Anwendungen auszutauschen. In diesem Muster sendet der Code, der einen Dienst aufruft, eine HTTP/HTTPS Anfrage an einen Endpunkt für den Dienst. Die Anforderung enthält die erforderlichen Parameter für einen erfolgreichen Aufruf der spezifischen Funktion. Der Service, der aufgerufen wird, generiert eine Antwort, die an den Anforderer zurückgesendet wird. Die Antwort enthält Daten, wenn die Operation erfolgreich war, oder Fehlerinformationen, wenn sie nicht erfolgreich war.

Das Aufrufen eines AWS Dienstes umfasst den gesamten Anforderungs- und Antwortzyklus eines Vorgangs an einem Dienstobjekt, einschließlich aller Wiederholungsversuche. Eine Anfrage enthält null oder mehr Eigenschaften als JSON-Parameter. Die Antwort ist in einem Objekt gekapselt, das sich auf den Vorgang bezieht, und wird über eine von mehreren Techniken, z. B. eine Rückruffunktion oder eine Zusage, an den Anforderer zurückgegeben. JavaScript

Themen

- [Serviceobjekte erstellen und aufrufen](#)
- [Rufen Sie Dienste asynchron auf](#)
- [Erstellen Sie Service-Client-Anfragen](#)
- [Bearbeiten Sie die Antworten der Servicekunden](#)
- [Arbeiten Sie mit JSON](#)
- [Aufrufe protokollieren AWS SDK für JavaScript](#)
- [Verwenden Sie AWS kontobasierte Endpunkte mit DynamoDB](#)
- [Schutz der Datenintegrität mit Amazon S3 S3-Prüfsummen](#)
- [SDK für JavaScript Codebeispiele](#)

Serviceobjekte erstellen und aufrufen

Die JavaScript API unterstützt die meisten verfügbaren AWS Dienste. Jeder Dienst in der JavaScript API stellt eine Clientklasse mit einer `send` Methode bereit, mit der Sie jede API aufrufen, die der Dienst unterstützt. Weitere Informationen zu Serviceklassen, Vorgängen und Parametern in der JavaScript API finden Sie in der [API-Referenz](#).

Wenn Sie das SDK in Node.js verwenden, fügen Sie Ihrer Anwendung das SDK-Paket für jeden Dienst hinzuimport, den Sie benötigen. Dadurch werden alle aktuellen Dienste unterstützt. Im folgenden Beispiel wird ein Amazon S3 S3-Serviceobjekt in der `us-west-1` Region erstellt.

```
// Import the Amazon S3 service client
import { S3Client } from "@aws-sdk/client-s3";
// Create an S3 client in the us-west-1 Region
const s3Client = new S3Client({
  region: "us-west-1"
});
```

Geben Sie die Parameter des Serviceobjekts an

Wenn Sie eine Methode eines Service-Objekts aufrufen, übergeben Sie die JSON-Parameter wie für die API erforderlich. Um beispielsweise in Amazon S3 ein Objekt für einen bestimmten Bucket und Schlüssel abzurufen, übergeben Sie der `GetObjectCommand` Methode die folgenden Parameter von `S3Client`. Weitere Informationen zum Übergeben von JSON-Parametern finden Sie unter [Arbeiten Sie mit JSON](#).

```
s3Client.send(new GetObjectCommand({Bucket: 'bucketName', Key: 'keyName'}));
```

Weitere Informationen zu Amazon S3 S3-Parametern finden Sie unter [@aws-sdk/client-s3](#) in der API-Referenz.

Verwenden Sie `@smithy/types` für generierte Clients in TypeScript

Wenn Sie das `@smithy/types` Paket verwenden TypeScript, können Sie die Eingabe- und Ausgabeformen eines Clients bearbeiten.

Szenario: **undefined** Aus Eingabe- und Ausgabestrukturen entfernen

Die Elemente der generierten Formen werden `undefined` für Eingabeformen mit vereinigt und sind `?` (optional) für Ausgabeformen verfügbar. Bei Eingaben wird dadurch die Validierung auf den

Dienst verschoben. Bei Ausgaben empfiehlt dies dringend, dass Sie die Ausgabedaten zur Laufzeit überprüfen sollten.

Wenn Sie diese Schritte überspringen möchten, verwenden Sie die `AssertiveClient` oder `UncheckedClient` geben Sie Helferlein ein. Im folgenden Beispiel werden die Typhelfer mit dem Amazon S3 S3-Service verwendet.

```
import { S3 } from "@aws-sdk/client-s3";
import type { AssertiveClient, UncheckedClient } from "@smithy/types";

const s3a = new S3({}) as AssertiveClient<S3>;
const s3b = new S3({}) as UncheckedClient<S3>;

// AssertiveClient enforces required inputs are not undefined
// and required outputs are not undefined.
const get = await s3a.getObject({
  Bucket: "",
  // @ts-expect-error (undefined not assignable to string)
  Key: undefined,
});

// UncheckedClient makes output fields non-nullable.
// You should still perform type checks as you deem
// necessary, but the SDK will no longer prompt you
// with nullability errors.
const body = await (
  await s3b.getObject({
    Bucket: "",
    Key: "",
  })
).Body.transformToString();
```

Wenn die Transformation auf einem nicht aggregierten Client mit der Command Syntax verwendet wird, kann die Eingabe nicht validiert werden, da sie eine andere Klasse durchläuft, wie im Beispiel unten gezeigt.

```
import { S3Client, ListBucketsCommand, GetObjectCommand, GetObjectCommandInput } from
"@aws-sdk/client-s3";
import type { AssertiveClient, UncheckedClient, NoUndefined } from "@smithy/types";

const s3 = new S3Client({}) as UncheckedClient<S3Client>;
```

```

const list = await s3.send(
  new ListBucketsCommand({
    // command inputs are not validated by the type transform.
    // because this is a separate class.
  })
);

/**
 * Although less ergonomic, you can use the NoUndefined<T>
 * transform on the input type.
 */
const getObjectInput: NoUndefined<GetObjectCommandInput> = {
  Bucket: "undefined",
  // @ts-expect-error (undefined not assignable to string)
  Key: undefined,
  // optional params can still be undefined.
  SSECustomerAlgorithm: undefined,
};

const get = s3.send(new GetObjectCommand(getObjectInput));

// outputs are still transformed.
await get.Body.TransformToString();

```

Szenario: Eingrenzung der Ausgaben-Nutzdaten-Blobtypen eines von Smithy TypeScript generierten Clients

Dieses Szenario ist vor allem für Operationen mit Streaming-Bodies relevant, z. B. innerhalb der Version 3. S3Client AWS SDK für JavaScript

Da die Blob-Payload-Typen plattformabhängig sind, sollten Sie in Ihrer Anwendung angeben, dass ein Client in einer bestimmten Umgebung ausgeführt wird. Dadurch werden die Blob-Nutzlasttypen eingegrenzt, wie im folgenden Beispiel gezeigt.

```

import { GetObjectCommand, S3Client } from "@aws-sdk/client-s3";
import type { NodeJsClient, SdkStream, StreamingBlobPayloadOutputTypes } from "@smithy/types";
import type { IncomingMessage } from "node:http";

// default client init.
const s3Default = new S3Client({});

```

```
// client init with type narrowing.
const s3NarrowType = new S3Client({}) as NodeJsClient<S3Client>;

// The default type of blob payloads is a wide union type including multiple possible
// request handlers.
const body1: StreamingBlobPayloadOutputTypes = (await s3Default.send(new
  GetObjectCommand({ Key: "", Bucket: "" })))
  .Body!;

// This is of the narrower type SdkStream<IncomingMessage> representing
// blob payload responses using specifically the node:http request handler.
const body2: SdkStream<IncomingMessage> = (await s3NarrowType.send(new
  GetObjectCommand({ Key: "", Bucket: "" })))
  .Body!;
```

Rufen Sie Dienste asynchron auf

Alle über das SDK ausgeführten Anforderungen sind asynchron. Dies ist wichtig, wenn Sie Browserskripts schreiben. JavaScript Die Ausführung in einem Webbrowser hat normalerweise nur einen einzigen Ausführungsthread. Nach einem asynchronen Aufruf eines AWS Dienstes wird das Browserskript weiter ausgeführt und kann dabei versuchen, Code auszuführen, der von diesem asynchronen Ergebnis abhängt, bevor er zurückkehrt.

Zu asynchronen Aufrufen eines AWS Dienstes gehört auch die Verwaltung dieser Aufrufe, sodass Ihr Code nicht versucht, Daten zu verwenden, bevor die Daten verfügbar sind. Die Themen in diesem Abschnitt erklären, wie wichtig es ist, asynchrone Aufrufe zu verwalten, und erläutern die verschiedenen Verwaltungstechniken im Detail.

Sie können zwar jede dieser Techniken verwenden, um asynchrone Aufrufe zu verwalten, wir empfehlen jedoch, sie `async/await` für jeden neuen Code zu verwenden.

`async/await`

Wir empfehlen Ihnen, diese Technik zu verwenden, da sie das Standardverhalten in Version 3 ist.

Verwenden Sie diese Technik in Browsern, die `Async/Await` nicht unterstützen.

Rückruf

Vermeiden Sie Rückrufe, außer in sehr einfachen Fällen. Sie könnten es jedoch für Migrationsszenarien nützlich finden.

Themen

- [Asynchrone Aufrufe verwalten](#)
- [Verwenden Sie async/await](#)
- [Verwenden Sie Versprechen JavaScript](#)
- [Verwenden Sie eine anonyme Rückruffunktion](#)

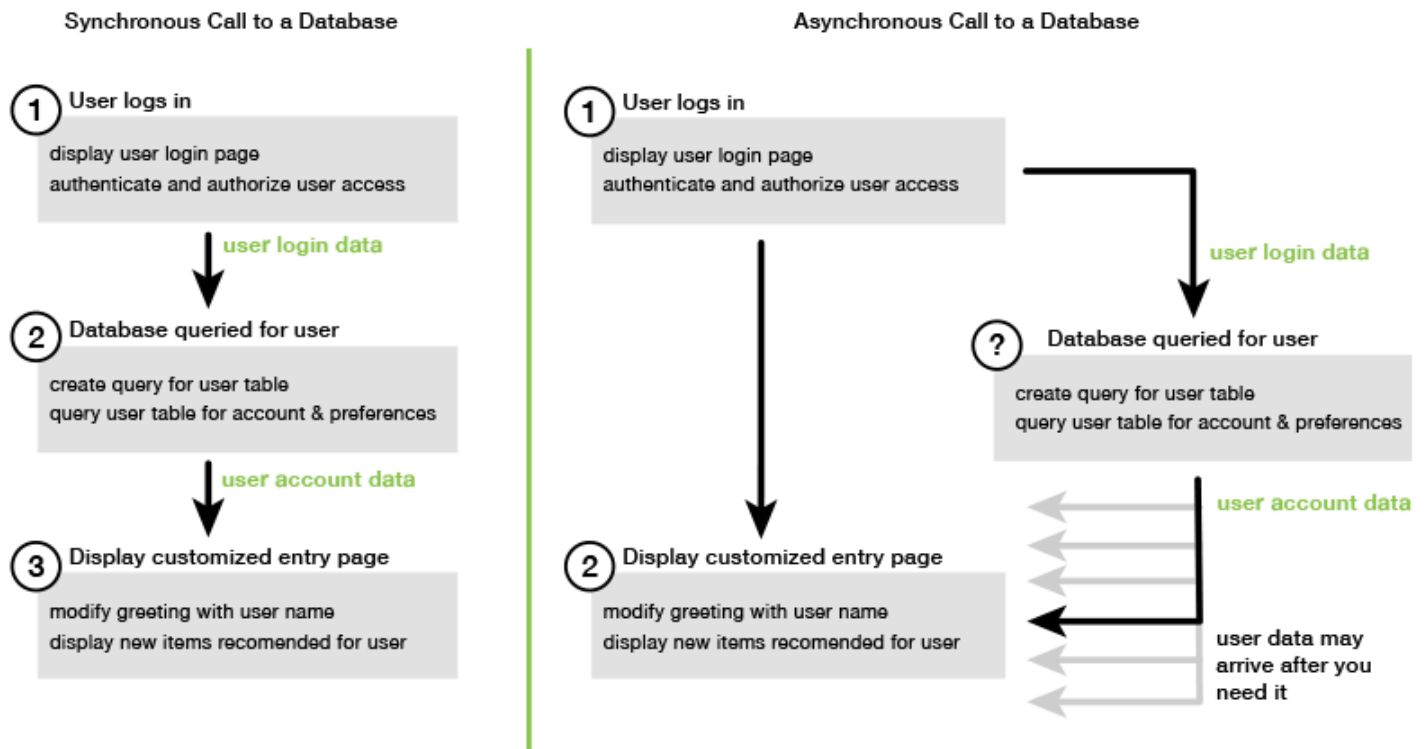
Asynchrone Aufrufe verwalten

Beispiel: Die Startseite einer E-Commerce-Website bietet wiederkehrenden Kunden die Möglichkeit, sich anzumelden. Die Kunden, die sich anmelden, profitieren u. a. davon, dass sich die Website nach der Anmeldung an ihre besonderen Präferenzen anpasst. Dafür ist Folgendes erforderlich:

1. Der Kunde muss sich anmelden und mit seinen Anmeldedaten validiert werden.
2. Die Präferenzen des Kunden werden von einer Kundendatenbank angefordert.
3. Die Datenbank stellt die Präferenzen des Kunden bereit, mit denen die Website angepasst wird, bevor die Seite geladen wird.

Wenn diese Aufgaben synchron ausgeführt werden, muss jede abgeschlossen sein, bevor die nächste starten kann. Das Laden der Webseite kann erst abgeschlossen werden, wenn die Kundeneinstellungen aus der Datenbank zurückgegeben werden. Nachdem die Datenbankabfrage an den Server gesendet wurde, kann der Empfang der Kundendaten jedoch aufgrund von Netzwerkengpässen, besonders hohem Datenverkehr oder einer schlechten Verbindung eines Mobilgeräts sich verzögern oder sogar fehlschlagen.

Um zu verhindern, dass die Website unter diesen Bedingungen einfriert, rufen Sie die Datenbank asynchron auf. Nachdem der Datenbankaufruf ausgeführt und Ihre asynchrone Anforderung gesendet wurde, wird Ihr Code wie erwartet weiter ausgeführt. Wenn Sie die Antwort eines asynchronen Aufrufs nicht ordnungsgemäß verwalten, versucht Ihr Code möglicherweise im Fall, dass diese Daten noch nicht verfügbar sind, Informationen zu verwenden, die er von der Datenbank zurückerwartet.



Verwenden Sie async/await

Anstatt Versprechen zu verwenden, sollten Sie `async/await` verwenden. Async-Funktionen sind einfacher und benötigen weniger Boilerplate als die Verwendung von Versprechen. `await` kann nur in einer asynchronen Funktion verwendet werden, um asynchron auf einen Wert zu warten.

Das folgende Beispiel verwendet `async/await`, um all Ihre Amazon DynamoDB-Tabellen in aufzulisten. `us-west-2`

Note

Damit dieses Beispiel ausgeführt werden kann:

- Installieren Sie den AWS SDK für JavaScript DynamoDB-Client, indem Sie ihn npm `install @aws-sdk/client-dynamodb` in die Befehlszeile Ihres Projekts eingeben.
- Stellen Sie sicher, dass Sie Ihre AWS Anmeldeinformationen korrekt konfiguriert haben. Weitere Informationen finden Sie unter [Legen Sie die Anmeldeinformationen fest](#).

```
import {
```

```
DynamoDBClient,  
ListTablesCommand  
} from "@aws-sdk/client-dynamodb";  
(async function () {  
  const dbClient = new DynamoDBClient({ region: "us-west-2" });  
  const command = new ListTablesCommand({});  
  
  try {  
    const results = await dbClient.send(command);  
    console.log(results.TableNames.join('\n'));  
  } catch (err) {  
    console.error(err)  
  }  
})();
```

Note

Nicht alle Browser unterstützen Async/Await. Eine Liste der Browser, die Unterstützung bieten, finden Sie unter [Async-Funktionen](#). async/await

Verwenden Sie Versprechen JavaScript

Verwenden Sie die AWS SDK für JavaScript v3-Methode (`ListTablesCommand`) des Service-Clients, um den Service aufzurufen und den asynchronen Ablauf zu verwalten, anstatt Callbacks zu verwenden. Das folgende Beispiel zeigt, wie Sie die Namen Ihrer Amazon DynamoDB-Tabellen abrufen können. `us-west-2`

```
import {  
  DynamoDBClient,  
  ListTablesCommand  
} from "@aws-sdk/client-dynamodb";  
const dbClient = new DynamoDBClient({ region: 'us-west-2' });  
  
dbClient.listtables(new ListTablesCommand({}))  
  .then(response => {  
    console.log(response.TableNames.join('\n'));  
  })  
  .catch((error) => {  
    console.error(error);  
  });
```

Koordinieren Sie mehrere Versprechen

In einigen Fällen muss Ihr Code mehrere asynchrone Aufrufe ausführen, die nur dann eine Aktion erfordern, wenn alle erfolgreich zurückgegeben wurden. Wenn Sie diese einzelnen asynchronen Methodenaufrufe mit Promises verwalten, können Sie ein zusätzliches Promise erstellen, das die `all`-Methode verwendet.

Diese Methode erfüllt dieses übergeordnete Promise-Objekt, falls und wenn das an die Methode übergebene Array der Promises erfüllt ist. Die Callback-Funktion wird als Array von Werten der Promises übergeben, die an die `all`-Methode übergeben werden.

Im folgenden Beispiel muss eine AWS Lambda Funktion drei asynchrone Aufrufe an Amazon DynamoDB tätigen, kann aber erst abgeschlossen werden, nachdem die Zusagen für jeden Aufruf erfüllt wurden.

```
const values = await Promise.all([firstPromise, secondPromise, thirdPromise]);

console.log("Value 0 is " + values[0].toString);
console.log("Value 1 is " + values[1].toString);
console.log("Value 2 is " + values[2].toString);

return values;
```

Browser- und Node.js unterstützen Promises

Die Support für native JavaScript Promises (ECMAScript 2015) hängt von der JavaScript Engine und Version ab, in der Ihr Code ausgeführt wird. Informationen zur Unterstützung von JavaScript Promises in den einzelnen Umgebungen, in denen Ihr Code ausgeführt werden muss, finden Sie in der [ECMAScript Kompatibilitätstabelle](#) unter GitHub.

Verwenden Sie eine anonyme Rückruffunktion

Jede Serviceobjektmethode kann eine anonyme Callback-Funktion als letzten Parameter akzeptieren. Die Signatur dieser Callback-Funktion lautet wie folgt.

```
function(error, data) {
    // callback handling code
};
```


Diese Callback-Funktion wird ausgeführt, wenn entweder eine erfolgreiche Antwort oder Fehlerdaten zurückgegeben werden. Wenn der Methodenaufruf erfolgreich war, wird der Callback-Funktion der Inhalt der Antwort im `data`-Parameter bereitgestellt. Wenn der Aufruf nicht erfolgreich war, werden die Details über den Fehler im `error`-Parameter angegeben.

In der Regel prüft der Code innerhalb der Callback-Funktion, ob ein Fehler vorliegt, den er verarbeitet, wenn ein Fehler zurückgegeben wird. Wenn kein Fehler zurückgegeben wird, ruft der Code die Antwortdaten vom `data`-Parameter ab. Die grundlegende Form der Callback-Funktion sieht wie folgt aus.

```
function(error, data) {  
  if (error) {  
    // error handling code  
    console.log(error);  
  } else {  
    // data handling code  
    console.log(data);  
  }  
};
```

Im vorherigen Beispiel werden die Details zum Fehler oder die zurückgegebenen Daten in der Konsole protokolliert. Hier finden Sie ein Beispiel für eine Callback-Funktion, die als Teil des Aufrufs einer Methode in einem Service-Objekt zurückgegeben wird.

```
ec2.describeInstances(function(error, data) {  
  if (error) {  
    console.log(error); // an error occurred  
  } else {  
    console.log(data); // request succeeded  
  }  
});
```

Erstellen Sie Service-Client-Anfragen

Das Stellen von Anfragen an AWS Serviceclients ist unkompliziert. Version 3 (V3) des SDK für JavaScript ermöglicht das Senden von Anfragen.

 Note

Sie können Operationen auch mit Befehlen der Version 2 (V2) ausführen, wenn Sie die Version V3 des SDK für verwenden JavaScript. Weitere Informationen finden Sie unter [Verwenden von v2-Befehlen](#).

Um eine Anfrage zu senden:

1. Initialisieren Sie ein Client-Objekt mit der gewünschten Konfiguration, z. B. einer bestimmten AWS Region.
2. (Optional) Erstellen Sie ein JSON-Anforderungsobjekt mit den Werten für die Anfrage, z. B. dem Namen eines bestimmten Amazon S3 S3-Buckets. Sie können die Parameter für die Anfrage überprüfen, indem Sie sich das API-Referenzthema für die Schnittstelle ansehen, deren Name der Client-Methode zugeordnet ist. Wenn Sie beispielsweise die *AbcCommand* Client-Methode verwenden, lautet die Anforderungsschnittstelle *AbcInput*.
3. Initialisieren Sie optional einen Dienstbefehl mit dem Anforderungsobjekt als Eingabe.
4. Rufen Sie send den Client mit dem Befehlsobjekt als Eingabe auf.

Um beispielsweise Ihre Amazon DynamoDB-Tabellen aufzulisten `us-west-2`, können Sie dies mit `async/await` tun.

```
import {
  DynamoDBClient,
  ListTablesCommand
} from "@aws-sdk/client-dynamodb";

(async function () {
  const dbClient = new DynamoDBClient({ region: 'us-west-2' });
  const command = new ListTablesCommand({});

  try {
    const results = await dbClient.send(command);
    console.log(results.TableNames.join('\n'));
  } catch (err) {
    console.error(err);
  }
})();
```

Bearbeiten Sie die Antworten der Servicekunden

Nachdem eine Dienstclient-Methode aufgerufen wurde, gibt sie eine Antwortobjektinstanz einer Schnittstelle zurück, deren Name der Client-Methode zugeordnet ist. Wenn Sie beispielsweise die `AbcCommand` Client-Methode verwenden, ist das Antwortobjekt vom Typ `AbcResponse` (Schnittstelle).

In der Antwort zurückgegebene Zugriffsdaten

Das Antwortobjekt enthält die Daten als Eigenschaften, die von der Serviceanfrage zurückgegeben wurden.

[Erstellen Sie Service-Client-Anfragen](#) In gab der `ListTablesCommand` Befehl die Tabellennamen in der `TableNames` Eigenschaft der Antwort zurück.

Auf Fehlerinformationen zugreifen

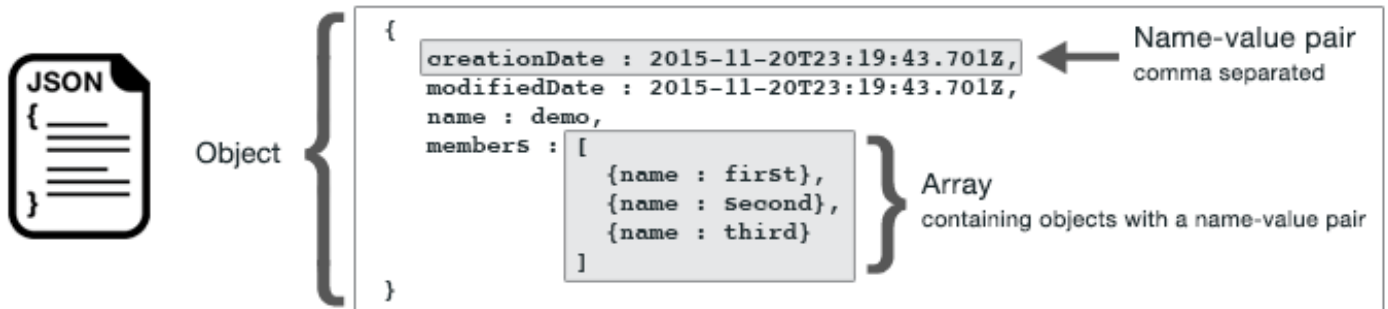
Wenn ein Befehl fehlschlägt, wird eine Ausnahme ausgelöst. Der folgende Codeausschnitt zeigt, wie eine Serviceausnahme behandelt werden kann.

```
try {
  await client.send(someCommand);
} catch (e) {
  if (e.name === "InvalidSignatureException") {
    // Handle InvalidSignatureException
  } else if (e.name === "ResourceNotFoundException") {
    // Handle ResourceNotFoundException
  } else if (e.name === "FooServiceException") {
    // Handle all other server-side exceptions from Foo service
  } else {
    // Handle errors from SDK
  }
}
```

Arbeiten Sie mit JSON

JSON ist ein Format für den Datenaustausch, das sowohl für Menschen als auch für Maschinen lesbar ist. Obwohl der Name JSON eine Abkürzung für JavaScript Object Notation ist, ist das Format von JSON unabhängig von jeder Programmiersprache.

The AWS SDK für JavaScript verwendet JSON, um bei Anfragen Daten an Serviceobjekte zu senden, und empfängt Daten von Serviceobjekten als JSON. Weitere Informationen über JSON finden Sie auf der Website json.org.



JSON stellt Daten auf zwei Arten dar:

- Als Objekt, das eine ungeordnete Sammlung von Name-Wert-Paaren ist. Ein Objekt wird innerhalb von zwei Klammern ({ und }) definiert. Jedes Name-Wert-Paar beginnt mit dem Namen, gefolgt von einem Doppelpunkt und dem Wert. Name-Wert-Paare werden durch Kommas voneinander getrennt.
- Als Array, das eine geordnete Sammlung von Werten ist. Ein Array wird innerhalb von zwei Klammern ([und]) definiert. Elemente im Array werden durch Kommas voneinander getrennt.

Hier sehen Sie ein Beispiel für ein JSON-Objekt mit einem Array von Objekten, in dem die Objekte Karten in einem Kartenspiel darstellen. Jede Karte wird durch zwei Namen-Wert-Paare definiert. Eines davon gibt einen eindeutigen Wert an, um die Karte zu identifizieren, und das andere gibt die URL an, die auf das Kartenbild verweist.

```
var cards = [
  {"CardID":"defaultname", "Image":"defaulturl"},
  {"CardID":"defaultname", "Image":"defaulturl"},
  {"CardID":"defaultname", "Image":"defaulturl"},
  {"CardID":"defaultname", "Image":"defaulturl"},
  {"CardID":"defaultname", "Image":"defaulturl"}
];
```

JSON als Serviceobjektparameter

Hier ist ein Beispiel für einfaches JSON, das verwendet wird, um die Parameter eines Aufrufs eines AWS Lambda Serviceobjekts zu definieren.

```
const params = {
  FunctionName : funcName,
  Payload : JSON.stringify(payload),
  LogType : LogType.Tail,
};
```

Das `params`-Objekt wird durch drei Name-Wert-Paare definiert, die durch Kommas innerhalb einer linken und einer rechten Klammer getrennt sind. Beim Bereitstellen von Parametern für einen Service-Objektmethodenaufruf werden die Namen durch die Parameternamen für die Service-Objektmethode bestimmt, die Sie aufrufen möchten. Beim Aufrufen einer Lambda-Funktion sind, und die Parameter `FunctionName`, `Payload`, die verwendet `LogType` werden, um die `invoke` Methode für ein Lambda-Serviceobjekt aufzurufen.

Wenn Sie Parameter an einen Methodenaufruf eines Serviceobjekts übergeben, stellen Sie das JSON-Objekt für den Methodenaufruf bereit, wie im folgenden Beispiel für den Aufruf einer Lambda-Funktion gezeigt.

```
const invoke = async (funcName, payload) => {
  const client = new LambdaClient({});
  const command = new InvokeCommand({
    FunctionName: funcName,
    Payload: JSON.stringify(payload),
    LogType: LogType.Tail,
  });

  const { Payload, LogResult } = await client.send(command);
  const result = Buffer.from(Payload).toString();
  const logs = Buffer.from(LogResult, "base64").toString();
  return { logs, result };
};
```

Aufrufe protokollieren AWS SDK für JavaScript

Das AWS SDK für JavaScript ist mit einem integrierten Logger ausgestattet, sodass Sie API-Aufrufe protokollieren können, die Sie mit dem SDK für tätigen. JavaScript

Um den Logger einzuschalten und Protokolleinträge in der Konsole zu drucken, konfigurieren Sie den Service-Client mit dem optionalen `logger` Parameter. Im folgenden Beispiel wird die Client-Protokollierung aktiviert, wobei Trace- und Debug-Ausgaben ignoriert werden.

```
new S3Client({
  logger: {
    ...console,
    debug(...args) {},
    trace(...args) {},
  },
});
```

Verwenden von Middleware zum Protokollieren von Anfragen

Der AWS SDK für JavaScript verwendet einen Middleware-Stack, um den Lebenszyklus eines Operationsaufrufs zu steuern. Jede Middleware im Stack ruft die nächste Middleware auf, nachdem sie Änderungen am Anforderungsobjekt vorgenommen hat. Dies erleichtert auch das Debuggen von Problemen im Stack erheblich, da Sie genau sehen können, welche Middleware aufgerufen wurde, bevor ein Fehler aufgetreten ist. Hier ist ein Beispiel für die Protokollierung von Anfragen mithilfe von Middleware:

```
const client = new DynamoDB({ region: "us-west-2" });

client.middlewareStack.add(
  (next, context) => async (args) => {
    console.log("AWS SDK context", context.clientName, context.commandName);
    console.log("AWS SDK request input", args.input);
    const result = await next(args);
    console.log("AWS SDK request output:", result.output);
    return result;
  },
  {
    name: "MyMiddleware",
    step: "build",
    override: true,
  }
);

await client.listTables({});
```

Im obigen Beispiel wird dem Middleware-Stack des DynamoDB-Clients eine Middleware hinzugefügt. Das erste Argument ist eine Funktion, die akzeptiert `next`, die nächste Middleware im Stack aufzurufen, und ein Objekt, das einige Informationen über die `context` aufgerufene Operation enthält. Es gibt eine Funktion zurück, die akzeptiert `args`, ein Objekt, das die an die Operation und

die Anforderung übergebenen Parameter enthält, und es gibt das Ergebnis des Aufrufs der nächsten Middleware mit zurück. `args`

Verwenden Sie AWS kontobasierte Endpunkte mit DynamoDB

DynamoDB bietet [AWS kontobasierte Endpunkte](#), die die Leistung verbessern können, indem sie Ihre AWS Konto-ID verwenden, um die Anforderungsweiterleitung zu optimieren.

Um diese Funktion nutzen zu können, müssen Sie Version 3.656.0 oder höher von Version 3 verwenden. AWS SDK für JavaScript Diese Funktion für kontobasierte Endgeräte ist in dieser neuen Version standardmäßig aktiviert.

Wenn Sie das kontobasierte Routing deaktivieren möchten, haben Sie die folgenden Optionen:

- Konfigurieren Sie einen DynamoDB-Dienstclient, dessen `accountIdEndpointMode` Parameter auf gesetzt ist. `disabled`
- Setzen Sie die Umgebungsvariable `AWS_ACCOUNT_ID_ENDPOINT_MODE` auf. `disabled`
- Aktualisieren Sie die Einstellung der gemeinsam genutzten AWS Konfigurationsdatei `account_id_endpoint_mode` auf `disabled`.

Der folgende Ausschnitt ist ein Beispiel dafür, wie Sie das kontobasierte Routing deaktivieren können, indem Sie einen DynamoDB-Dienstclient konfigurieren:

```
const ddbClient = new DynamoDBClient({
  region: "us-west-2",
  accountIdEndpointMode: "disabled" // Disable account ID in the endpoint
});
```

[Das AWS SDKs Referenzhandbuch zu Tools enthält weitere Informationen zu den anderen Konfigurationsoptionen.](#)

Schutz der Datenintegrität mit Amazon S3 S3-Prüfsummen

Amazon Simple Storage Service (Amazon S3) bietet die Möglichkeit, beim Hochladen eines Objekts eine Prüfsumme anzugeben. Wenn Sie eine Prüfsumme angeben, wird diese zusammen mit dem Objekt gespeichert und kann beim Herunterladen des Objekts überprüft werden.

Prüfsummen bieten eine zusätzliche Ebene der Datenintegrität bei der Übertragung von Dateien. Mit Prüfsummen können Sie die Datenkonsistenz überprüfen, indem Sie sicherstellen, dass die

empfangene Datei mit der Originaldatei übereinstimmt. Weitere Informationen zu Prüfsummen mit Amazon S3 finden Sie im [Amazon Simple Storage Service-Benutzerhandbuch](#), einschließlich der [unterstützten Algorithmen](#).

Sie haben die Flexibilität, den Algorithmus auszuwählen, der Ihren Anforderungen am besten entspricht, und das SDK die Prüfsumme berechnen zu lassen. Alternativ können Sie mithilfe eines der unterstützten Algorithmen einen vorab berechneten Prüfsummenwert angeben.

Note

Ab Version 3.729.0 von bietet das SDK standardmäßige Integritätsschutzmaßnahmen AWS SDK für JavaScript, indem es automatisch eine Prüfsumme für Uploads berechnet. CRC32 Das SDK berechnet diese Prüfsumme, wenn Sie keinen vorab berechneten Prüfsummenwert angeben oder wenn Sie keinen Algorithmus angeben, den das SDK zur Berechnung einer Prüfsumme verwenden soll.

[Das SDK bietet auch globale Einstellungen für den Schutz der Datenintegrität, die Sie extern festlegen können. Weitere Informationen finden Sie im Referenzhandbuch und im Tools-Referenzhandbuch.AWS SDKs](#)

Hochladen eines Objekts

Sie laden Objekte auf Amazon S3 hoch, indem Sie den [PutObject](#)Befehl von verwendenS3Client. Verwenden Sie den ChecksumAlgorithm Parameter des Builders für, PutObjectRequest um die Prüfsummenberechnung zu aktivieren und den Algorithmus anzugeben. Gültige Werte finden Sie [unter Unterstützte Prüfsummenalgorithmen](#).

Der folgende Codeausschnitt zeigt eine Anfrage zum Hochladen eines Objekts mit einer CRC-32-Prüfsumme. Wenn das SDK die Anfrage sendet, berechnet es die CRC-32-Prüfsumme und lädt das Objekt hoch. Amazon S3 speichert die Prüfsumme mit dem Objekt.

```
import { ChecksumAlgorithm, S3 } from "@aws-sdk/client-s3";

const client = new S3();
const response = await client.putObject({
  Bucket: "my-bucket",
  Key: "my-key",
  Body: "Hello, world!",
  ChecksumAlgorithm: ChecksumAlgorithm.CRC32,
```



```
});
```

Wenn Sie mit der Anfrage keinen Prüfsummenalgorithmus angeben, variiert das Prüfsummenverhalten je nach der Version des verwendeten SDK, wie in der folgenden Tabelle dargestellt.

Prüfsummenverhalten, wenn kein Prüfsummenalgorithmus bereitgestellt wird

SDK für Version JavaScript	Verhalten der Prüfsumme
Vor 3.729.0	Das SDK berechnet nicht automatisch eine CRC-basierte Prüfsumme und gibt sie in der Anfrage an.
3.729.0 oder höher	Das SDK verwendet den CRC32 Algorithmus zur Berechnung der Prüfsumme und stellt sie in der Anfrage bereit. Amazon S3 validiert die Integrität der Übertragung, indem es seine eigene CRC32 Prüfsumme berechnet und sie mit der vom SDK bereitgestellten Prüfsumme vergleicht. Wenn die Prüfsummen übereinstimmen, wird die Prüfsumme zusammen mit dem Objekt gespeichert.

Wenn die vom SDK berechnete Prüfsumme nicht mit der Prüfsumme übereinstimmt, die Amazon S3 beim Empfang der Anfrage berechnet, wird ein Fehler zurückgegeben.

Verwenden Sie einen vorberechneten Prüfsummenwert

Ein mit der Anfrage bereitgestellter vorberechneter Prüfsummenwert deaktiviert die automatische Berechnung durch das SDK und verwendet stattdessen den angegebenen Wert.

Das folgende Beispiel zeigt eine Anfrage mit einer vorberechneten SHA-256-Prüfsumme.

```
import { S3 } from "@aws-sdk/client-s3";
import { createHash } from "node:crypto";

const client = new S3();
```

```
const Body = "Hello, world!";
const ChecksumSHA256 = await createHash("sha256").update(Body).digest("base64");

const response = await client.putObject({
  Bucket: "my-bucket",
  Key: "my-key",
  Body,
  ChecksumSHA256,
});
```

Wenn Amazon S3 feststellt, dass der Prüfsummenwert für den angegebenen Algorithmus falsch ist, gibt der Service eine Fehlerantwort zurück.

Mehrteilige Uploads

Sie können Prüfsummen auch bei mehrteiligen Uploads verwenden. Sie AWS SDK für JavaScript können die Upload Bibliotheksoptionen von bis zur Verwendung von Prüfsummen bei `@aws-sdk/lib-storage` mehrteiligen Uploads verwenden.

```
import { ChecksumAlgorithm, S3 } from "@aws-sdk/client-s3";
import { Upload } from "@aws-sdk/lib-storage";
import { createReadStream } from "node:fs";

const client = new S3();
const filePath = "/path/to/file";
const Body = createReadStream(filePath);

const upload = new Upload({
  client,
  params: {
    Bucket: "my-bucket",
    Key: "my-key",
    Body,
    ChecksumAlgorithm: ChecksumAlgorithm.CRC32,
  },
});
await upload.done();
```

SDK für JavaScript Codebeispiele

Die Themen in diesem Abschnitt enthalten Beispiele für die AWS SDK für JavaScript Verwendung APIs verschiedener Dienste zur Ausführung allgemeiner Aufgaben.

Den Quellcode für diese und andere Beispiele finden Sie im [AWS Codebeispiel-Repository unter GitHub](#). Um dem AWS Dokumentationsteam ein neues Codebeispiel vorzuschlagen, das es erstellen könnte, erstellen Sie eine Anfrage. Das Team möchte Codebeispiele erstellen, die breitere Szenarien und Anwendungsfälle abdecken, im Vergleich zu einfachen Codeausschnitten, die nur einzelne API-Aufrufe abdecken. Eine Anleitung dazu findest du im Abschnitt „Code erstellen“ in den [Richtlinien für Beiträge](#) von GitHub.

Important

In diesen Beispielen wird die ECMAScript6 Import-/Export-Syntax verwendet.

- Dazu ist die Version 14.17 von Node.js oder höher erforderlich. Informationen zum Herunterladen und Installieren der neuesten Version von Node.js finden Sie unter [Node.js downloads](#).
- Wenn Sie die CommonJS-Syntax bevorzugen, finden Sie [JavaScript ES6/CommonJS-Syntax](#) die Konvertierungsrichtlinien.

Themen

- [JavaScript ES6/CommonJs-Syntax](#)
- [AWS Elemental MediaConvert Beispiele](#)
- [AWS Lambda Beispiele](#)
- [Amazon-Lex-Beispiele](#)
- [Beispiele für Amazon Polly](#)
- [Beispiele für Amazon Redshift](#)
- [Beispiele für Amazon Simple Email Service](#)
- [Beispiele für Amazon Simple Notification Service](#)
- [Beispiele für Amazon Transcribe](#)
- [Node.js auf einer Amazon EC2 EC2-Instance einrichten](#)
- [Lambda mit API Gateway aufrufen](#)
- [Geplante Ereignisse zur Ausführung von AWS Lambda Funktionen erstellen](#)
- [Erstellen eines Amazon-Lex-Chatbots](#)

JavaScript ES6/CommonJs-Syntax

Die AWS SDK für JavaScript Codebeispiele sind in ECMAScript 6 (ES6) geschrieben. ES6 bietet neue Syntax und neue Funktionen, um Ihren Code moderner und lesbarer zu machen und mehr zu erreichen.

ES6 erfordert, dass Sie Node.js Version 13.x oder höher verwenden. Informationen zum Herunterladen und Installieren der neuesten Version von Node.js finden Sie unter [Node.js downloads](#). Wenn Sie es vorziehen, können Sie jedoch jedes unserer Beispiele anhand der folgenden Richtlinien in die CommonJS-Syntax konvertieren:

- Entfernen Sie es "type" : "module" aus der package.json in Ihrer Projektumgebung.
- Konvertiert alle ES6 import Anweisungen in require CommonJS-Anweisungen. Konvertiere zum Beispiel:

```
import { CreateBucketCommand } from "@aws-sdk/client-s3";
import { s3 } from "../libs/s3Client.js";
```

Zu seinem CommonJS-Äquivalent:

```
const { CreateBucketCommand } = require("@aws-sdk/client-s3");
const { s3 } = require("../libs/s3Client.js");
```

- Konvertiert alle ES6 export Anweisungen in module.exports CommonJS-Anweisungen. Konvertiere zum Beispiel:

```
export {s3}
```

Zu seinem CommonJS-Äquivalent:

```
module.exports = {s3}
```

Das folgende Beispiel zeigt das Codebeispiel für die Erstellung eines Amazon S3 S3-Buckets ES6 sowohl in CommonJS als auch in CommonJS.

ES6

libs/s3Client.js

```
// Create service client module using ES6 syntax.
import { S3Client } from "@aws-sdk/client-s3";
// Set the AWS region
const REGION = "eu-west-1"; //e.g. "us-east-1"
// Create Amazon S3 service object.
const s3 = new S3Client({ region: REGION });
// Export 's3' constant.
export {s3};
```

s3_createbucket.js

```
// Get service clients module and commands using ES6 syntax.
import { CreateBucketCommand } from "@aws-sdk/client-s3";
import { s3 } from "../libs/s3Client.js";

// Get service clients module and commands using CommonJS syntax.
// const { CreateBucketCommand } = require("@aws-sdk/client-s3");
// const { s3 } = require("../libs/s3Client.js");

// Set the bucket parameters
const bucketParams = { Bucket: "BUCKET_NAME" };

// Create the Amazon S3 bucket.
const run = async () => {
  try {
    const data = await s3.send(new CreateBucketCommand(bucketParams));
    console.log("Success", data.Location);
    return data;
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

CommonJS

libs/s3Client.js

```
// Create service client module using CommonJS syntax.
const { S3Client } = require("@aws-sdk/client-s3");
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create Amazon S3 service object.
const s3 = new S3Client({ region: REGION });
// Export 's3' constant.
module.exports = {s3};
```

s3_createbucket.js

```
// Get service clients module and commands using CommonJS syntax.
const { CreateBucketCommand } = require("@aws-sdk/client-s3");
const { s3 } = require("../libs/s3Client.js");

// Set the bucket parameters
const bucketParams = { Bucket: "BUCKET_NAME" };

// Create the Amazon S3 bucket.
const run = async () => {
  try {
    const data = await s3.send(new CreateBucketCommand(bucketParams));
    console.log("Success", data.Location);
    return data;
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

AWS Elemental MediaConvert Beispiele

AWS Elemental MediaConvert ist ein dateibasierter Videotranskodierungsdienst mit Funktionen in Broadcast-Qualität. Sie können damit Inhalte für die Übertragung und für die video-on-demand (VOD-) Übertragung über das Internet erstellen. Weitere Informationen finden Sie im [AWS Elemental MediaConvert -Benutzerhandbuch](#).

Die JavaScript API für MediaConvert wird über die MediaConvert Client-Klasse verfügbar gemacht. Weitere Informationen finden Sie unter [Class: MediaConvert](#) in der API-Referenz.

Themen

- [Transcodierungsaufträge erstellen und verwalten in MediaConvert](#)
- [Verwenden von Jobvorlagen in MediaConvert](#)

Transcodierungsaufträge erstellen und verwalten in MediaConvert



Dieses Node.js-Codebeispiel zeigt:

- So erstellen Sie Transcodierungsaufträge in MediaConvert
- So stornieren Sie einen Transcodierungsauftrag.
- So rufen Sie die JSON für einen abgeschlossenen Transcodierungsauftrag ab.
- So rufen Sie ein JSON-Array für bis zu 20 der zuletzt erstellten Aufträge ab.

Das Szenario

In diesem Beispiel verwenden Sie ein Modul Node.js zum Aufrufen, um Transcodierungsaufträge MediaConvert zu erstellen und zu verwalten. Der Code verwendet JavaScript dazu das SDK, indem er die folgenden Methoden der MediaConvert Client-Klasse verwendet:

- [CreateJobCommand](#)
- [CancelJobCommand](#)
- [GetJobCommand](#)
- [ListJobsCommand](#)

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels schließen Sie zunächst diese Aufgaben ab:

- Richten Sie die Projektumgebung ein, um diese TypeScript Node-Beispiele auszuführen, und installieren Sie die erforderlichen Module AWS SDK für JavaScript und Module von Drittanbietern. Folgen Sie den Anweisungen auf [GitHub](#).
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zur Bereitstellung einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter [Dateien mit gemeinsam genutzten Konfigurationen und Anmeldeinformationen](#) im AWS SDKs Referenzhandbuch zu Tools.
- Erstellen und konfigurieren Sie Amazon S3 S3-Buckets, die Speicherplatz für Auftragseingabedateien und Ausgabedateien bereitstellen. Einzelheiten finden Sie unter [Speicher für Dateien erstellen](#) im AWS Elemental MediaConvert Benutzerhandbuch.
- Laden Sie das Eingabevideo in den Amazon S3 S3-Bucket hoch, den Sie für den Eingabespeicher bereitgestellt haben. Eine Liste der unterstützten Eingabe-Videocodecs und Container finden Sie im Benutzerhandbuch unter [Unterstützte Eingabecodecs und Container](#). AWS Elemental MediaConvert
- Erstellen Sie eine IAM-Rolle, die MediaConvert Zugriff auf Ihre Eingabedateien und die Amazon S3 S3-Buckets gewährt, in denen Ihre Ausgabedateien gespeichert sind. Einzelheiten finden Sie unter [Einrichten von IAM-Berechtigungen](#) im AWS Elemental MediaConvert Benutzerhandbuch.

 Important

In diesem Beispiel wird ECMAScript6 (ES6) verwendet. Dies erfordert Node.js Version 13.x oder höher. Informationen zum Herunterladen und Installieren der neuesten Version von Node.js finden Sie unter [Node.js downloads](#). .

Wenn Sie jedoch die CommonJS-Syntax bevorzugen, finden Sie weitere Informationen unter [JavaScript ES6/CommonJs-Syntax](#)

Definition eines einfachen Transcodierungsauftrags

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `emc_createjob.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor gezeigt konfigurieren, einschließlich der Installation der erforderlichen Clients und Pakete. Erstellen Sie die JSON für die Definition der Transcodierungsauftragsparameter.

Diese Parameter sind sehr detailliert. Sie können die [AWS Elemental MediaConvert Konsole](#) verwenden, um die JSON-Jobparameter zu generieren, indem Sie Ihre Job-Einstellungen in der

Konsole auswählen und dann am Ende des Job-Abschnitts die Option Job-JSON anzeigen wählen. Dieses Beispiel enthält die JSON für einen einfachen Auftrag.

Note

JOB_QUEUE_ARN Ersetzen Sie durch die MediaConvert Job-Warteschlange, ***IAM_ROLE_ARN*** durch den Amazon-Ressourcennamen (ARN) der IAM-Rolle, ***OUTPUT_BUCKET_NAME*** durch den Ziel-Bucket-Namen — zum Beispiel "s3://OUTPUT_BUCKET_NAME/", und ***INPUT_BUCKET_AND_FILENAME*** durch den Eingabe-Bucket und den Dateinamen — zum Beispiel "s3://INPUT_BUCKET/FILE_NAME".

```
const params = {
  Queue: "JOB_QUEUE_ARN", //JOB_QUEUE_ARN
  UserMetadata: {
    Customer: "Amazon",
  },
  Role: "IAM_ROLE_ARN", //IAM_ROLE_ARN
  Settings: {
    OutputGroups: [
      {
        Name: "File Group",
        OutputGroupSettings: {
          Type: "FILE_GROUP_SETTINGS",
          FileGroupSettings: {
            Destination: "OUTPUT_BUCKET_NAME", //OUTPUT_BUCKET_NAME, e.g., "s3://
            BUCKET_NAME/"
          },
        },
      },
    ],
    Outputs: [
      {
        VideoDescription: {
          ScalingBehavior: "DEFAULT",
          TimecodeInsertion: "DISABLED",
          AntiAlias: "ENABLED",
          Sharpness: 50,
          CodecSettings: {
            Codec: "H_264",
            H264Settings: {
              InterlaceMode: "PROGRESSIVE",
              NumberReferenceFrames: 3,
              Syntax: "DEFAULT",
            },
          },
        },
      },
    ],
  },
}
```

```

        Softness: 0,
        GopClosedCadence: 1,
        GopSize: 90,
        Slices: 1,
        GopBReference: "DISABLED",
        SlowPal: "DISABLED",
        SpatialAdaptiveQuantization: "ENABLED",
        TemporalAdaptiveQuantization: "ENABLED",
        FlickerAdaptiveQuantization: "DISABLED",
        EntropyEncoding: "CABAC",
        Bitrate: 5000000,
        FramerateControl: "SPECIFIED",
        RateControlMode: "CBR",
        CodecProfile: "MAIN",
        Telecine: "NONE",
        MinIInterval: 0,
        AdaptiveQuantization: "HIGH",
        CodecLevel: "AUTO",
        FieldEncoding: "PAFF",
        SceneChangeDetect: "ENABLED",
        QualityTuningLevel: "SINGLE_PASS",
        FramerateConversionAlgorithm: "DUPLICATE_DROP",
        UnregisteredSeiTimecode: "DISABLED",
        GopSizeUnits: "FRAMES",
        ParControl: "SPECIFIED",
        NumberBFramesBetweenReferenceFrames: 2,
        RepeatPps: "DISABLED",
        FramerateNumerator: 30,
        FramerateDenominator: 1,
        ParNumerator: 1,
        ParDenominator: 1,
    },
},
AfdSignaling: "NONE",
DropFrameTimecode: "ENABLED",
RespondToAfd: "NONE",
ColorMetadata: "INSERT",
},
AudioDescriptions: [
    {
        AudioTypeControl: "FOLLOW_INPUT",
        CodecSettings: {
            Codec: "AAC",
            AacSettings: {

```

```

        AudioDescriptionBroadcasterMix: "NORMAL",
        RateControlMode: "CBR",
        CodecProfile: "LC",
        CodingMode: "CODING_MODE_2_0",
        RawFormat: "NONE",
        SampleRate: 48000,
        Specification: "MPEG4",
        Bitrate: 64000,
    },
},
    LanguageCodeControl: "FOLLOW_INPUT",
    AudioSourceName: "Audio Selector 1",
},
],
ContainerSettings: {
    Container: "MP4",
    Mp4Settings: {
        CslgAtom: "INCLUDE",
        FreeSpaceBox: "EXCLUDE",
        MoovPlacement: "PROGRESSIVE_DOWNLOAD",
    },
},
    NameModifier: "_1",
},
],
},
],
AdAvailOffset: 0,
Inputs: [
    {
        AudioSelectors: {
            "Audio Selector 1": {
                Offset: 0,
                DefaultSelection: "NOT_DEFAULT",
                ProgramSelection: 1,
                SelectorType: "TRACK",
                Tracks: [1],
            },
        },
        VideoSelector: {
            ColorSpace: "FOLLOW",
        },
        FilterEnable: "AUTO",
        PsiControl: "USE_PSI",
    }
]

```

```

    FilterStrength: 0,
    DeblockFilter: "DISABLED",
    DenoiseFilter: "DISABLED",
    TimecodeSource: "EMBEDDED",
    FileInput: "INPUT_BUCKET_AND_FILENAME", //INPUT_BUCKET_AND_FILENAME, e.g.,
"s3://BUCKET_NAME/FILE_NAME"
  },
],
TimecodeConfig: {
  Source: "EMBEDDED",
},
},
};

```

Einen Transcodierungsauftrag erstellen

Rufen Sie nach dem Erstellen der JSON-Jobparameter die asynchrone `run` Methode auf, um ein `MediaConvert Client`-Dienstobjekt aufzurufen, und übergeben Sie dabei die Parameter. Die ID des erstellten Auftrags wird in den `data` der Antwort zurückgegeben.

```

const run = async () => {
  try {
    const data = await emcClient.send(new CreateJobCommand(params));
    console.log("Job created!", data);
    return data;
  } catch (err) {
    console.log("Error", err);
  }
};
run();

```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node emc_createjob.js
```

Den vollständigen Beispielcode finden Sie [hier unter GitHub](#).

Abbrechen eines Transcodierungsauftrags

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `emc_canceljob.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor gezeigt konfigurieren, einschließlich des Herunterladens der erforderlichen

Clients und Pakete. Erstellen Sie die JSON, die die ID des zu stornierenden Auftrags enthält. Rufen Sie dann die `CancelJobCommand` Methode auf, indem Sie ein Versprechen zum Aufrufen eines `MediaConvert` Client-Dienstobjekts erstellen und die Parameter übergeben. Verarbeiten Sie die Antwort im Promise-Rückruf.

 Note

JOB_ID Ersetzen Sie es durch die ID des Jobs, der abgebrochen werden soll.

```
// Import required AWS-SDK clients and commands for Node.js
import { CancelJobCommand } from "@aws-sdk/client-mediaconvert";
import { emcClient } from "../libs/emcClient.js";

// Set the parameters
const params = { Id: "JOB_ID" }; //JOB_ID

const run = async () => {
  try {
    const data = await emcClient.send(new CancelJobCommand(params));
    console.log(`Job ${params.Id} is canceled`);
    return data;
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Um das Beispiel auszuführen, geben Sie in der Befehlszeile Folgendes ein.

```
node ec2_canceljob.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Liste der letzten Transcodierungsaufträge

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `emc_listjobs.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor gezeigt konfigurieren, einschließlich der Installation der erforderlichen Clients und Pakete.

Erstellen Sie die JSON-Parameter, einschließlich Werten, um anzugeben ASCENDING, ob DESCENDING die Liste nach dem Amazon-Ressourcennamen (ARN) der zu prüfenden Job-Warteschlange und dem Status der einzuschließenden Jobs sortiert werden soll. Rufen Sie dann die `ListJobsCommand` Methode auf, indem Sie eine Zusage für den Aufruf eines `MediaConvert` Client-Serviceobjekts erstellen und die Parameter übergeben.

 Note

`QUEUE_ARN` Ersetzen Sie durch den Amazon-Ressourcennamen (ARN) der zu prüfenden Auftragswarteschlange und `STATUS` durch den Status der Warteschlange.

```
// Import required AWS-SDK clients and commands for Node.js
import { ListJobsCommand } from "@aws-sdk/client-mediaconvert";
import { emcClient } from "../libs/emcClient.js";

// Set the parameters
const params = {
  MaxResults: 10,
  Order: "ASCENDING",
  Queue: "QUEUE_ARN",
  Status: "SUBMITTED", // e.g., "SUBMITTED"
};

const run = async () => {
  try {
    const data = await emcClient.send(new ListJobsCommand(params));
    console.log("Success. Jobs: ", data.Jobs);
  } catch (err) {
    console.log("Error", err);
  }
};

run();
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node emc_listjobs.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Verwenden von Jobvorlagen in MediaConvert



Dieses Node.js-Codebeispiel zeigt:

- So erstellen Sie AWS Elemental MediaConvert Jobvorlagen.
- So verwenden Sie eine Auftragsvorlage zum Erstellen eines Transcodierungsauftrags.
- So listen Sie alle Ihre Auftragsvorlagen auf.
- So löschen Sie Auftragsvorlagen.

Das Szenario

Die JSON-Datei, die für die Erstellung eines Transcodierungsauftrags erforderlich MediaConvert ist, ist detailliert und enthält eine große Anzahl von Einstellungen. Sie können die Auftragserstellung erheblich vereinfachen, indem Sie zweifelsfrei funktionierende Einstellungen in einer Auftragsvorlage speichern, die zum Erstellen nachfolgender Aufträge verwendet werden kann. In diesem Beispiel verwenden Sie ein Modul Node.js zum Aufrufen, um Jobvorlagen MediaConvert zu erstellen, zu verwenden und zu verwalten. Der Code verwendet JavaScript dazu das SDK, indem er die folgenden Methoden der MediaConvert Client-Klasse verwendet:

- [CreateJobTemplateCommand](#)
- [CreateJobCommand](#)
- [DeleteJobTemplateCommand](#)
- [ListJobTemplatesCommand](#)

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels schließen Sie zunächst diese Aufgaben ab:

- Richten Sie die Projektumgebung ein, um diese TypeScript Node-Beispiele auszuführen, und installieren Sie die erforderlichen Module AWS SDK für JavaScript und Module von Drittanbietern. Folgen Sie den Anweisungen auf [GitHub](#).

- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zur Bereitstellung einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter [Dateien mit gemeinsam genutzten Konfigurationen und Anmeldeinformationen](#) im AWS SDKs Referenzhandbuch zu Tools.
- Erstellen Sie eine IAM-Rolle, die MediaConvert Zugriff auf Ihre Eingabedateien und die Amazon S3 S3-Buckets gewährt, in denen Ihre Ausgabedateien gespeichert sind. Einzelheiten finden Sie unter [Einrichten von IAM-Berechtigungen](#) im AWS Elemental MediaConvert Benutzerhandbuch.

Important

In diesen Beispielen wird ECMAScript6 (ES6) verwendet. Dies erfordert Node.js Version 13.x oder höher. Informationen zum Herunterladen und Installieren der neuesten Version von Node.js finden Sie unter [Node.js downloads](#).

Wenn Sie jedoch die CommonJS-Syntax bevorzugen, finden Sie weitere Informationen unter [JavaScript ES6/CommonJs-Syntax](#).

Eine Jobvorlage erstellen

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `emc_create_jobtemplate.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor gezeigt konfigurieren, einschließlich der Installation der erforderlichen Clients und Pakete.

Geben Sie die Parameter-JSON für die Vorlagenerstellung an. Sie können die meisten JSON-Parameter aus einem vorherigen erfolgreichen Job verwenden, um die Settings-Werte in der Vorlage anzugeben. In diesem Beispiel werden die Aufgabeneinstellungen aus [Transcodierungsaufträge erstellen und verwalten in MediaConvert](#) verwendet.

Rufen Sie die `CreateJobTemplateCommand` Methode auf, indem Sie ein Versprechen für den Aufruf eines MediaConvert Client-Dienstobjekts erstellen und die Parameter übergeben.

Note

`JOB_QUEUE_ARN` Ersetzen Sie es durch den Amazon-Ressourcennamen (ARN) der zu prüfenden Auftragswarteschlange und **`BUCKET_NAME`** durch den Namen des Amazon S3-Ziel-Buckets, zum Beispiel `"s3://BUCKET_NAME/"`.


```
// Import required AWS-SDK clients and commands for Node.js
import { CreateJobTemplateCommand } from "@aws-sdk/client-mediaconvert";
import { emcClient } from "../libs/emcClient.js";

const params = {
  Category: "YouTube Jobs",
  Description: "Final production transcode",
  Name: "DemoTemplate",
  Queue: "JOB_QUEUE_ARN", //JOB_QUEUE_ARN
  Settings: {
    OutputGroups: [
      {
        Name: "File Group",
        OutputGroupSettings: {
          Type: "FILE_GROUP_SETTINGS",
          FileGroupSettings: {
            Destination: "BUCKET_NAME", // BUCKET_NAME e.g., "s3://BUCKET_NAME/"
          },
        },
      },
    ],
    Outputs: [
      {
        VideoDescription: {
          ScalingBehavior: "DEFAULT",
          TimecodeInsertion: "DISABLED",
          AntiAlias: "ENABLED",
          Sharpness: 50,
          CodecSettings: {
            Codec: "H_264",
            H264Settings: {
              InterlaceMode: "PROGRESSIVE",
              NumberReferenceFrames: 3,
              Syntax: "DEFAULT",
              Softness: 0,
              GopClosedCadence: 1,
              GopSize: 90,
              Slices: 1,
              GopBReference: "DISABLED",
              SlowPal: "DISABLED",
              SpatialAdaptiveQuantization: "ENABLED",
              TemporalAdaptiveQuantization: "ENABLED",
              FlickerAdaptiveQuantization: "DISABLED",
              EntropyEncoding: "CABAC",
              Bitrate: 5000000,
            },
          },
        },
      },
    ],
  },
}
```

```

    FramerateControl: "SPECIFIED",
    RateControlMode: "CBR",
    CodecProfile: "MAIN",
    Telecine: "NONE",
    MinIInterval: 0,
    AdaptiveQuantization: "HIGH",
    CodecLevel: "AUTO",
    FieldEncoding: "PAFF",
    SceneChangeDetect: "ENABLED",
    QualityTuningLevel: "SINGLE_PASS",
    FramerateConversionAlgorithm: "DUPLICATE_DROP",
    UnregisteredSeiTimecode: "DISABLED",
    GopSizeUnits: "FRAMES",
    ParControl: "SPECIFIED",
    NumberBFramesBetweenReferenceFrames: 2,
    RepeatPps: "DISABLED",
    FramerateNumerator: 30,
    FramerateDenominator: 1,
    ParNumerator: 1,
    ParDenominator: 1,
  },
},
AfdSignaling: "NONE",
DropFrameTimecode: "ENABLED",
RespondToAfd: "NONE",
ColorMetadata: "INSERT",
},
AudioDescriptions: [
  {
    AudioTypeControl: "FOLLOW_INPUT",
    CodecSettings: {
      Codec: "AAC",
      AacSettings: {
        AudioDescriptionBroadcasterMix: "NORMAL",
        RateControlMode: "CBR",
        CodecProfile: "LC",
        CodingMode: "CODING_MODE_2_0",
        RawFormat: "NONE",
        SampleRate: 48000,
        Specification: "MPEG4",
        Bitrate: 64000,
      },
    },
  },
],
LanguageCodeControl: "FOLLOW_INPUT",

```

```
        AudioSourceName: "Audio Selector 1",
      },
    ],
    ContainerSettings: {
      Container: "MP4",
      Mp4Settings: {
        CslgAtom: "INCLUDE",
        FreeSpaceBox: "EXCLUDE",
        MoovPlacement: "PROGRESSIVE_DOWNLOAD",
      },
    },
    NameModifier: "_1",
  },
],
AdAvailOffset: 0,
Inputs: [
  {
    AudioSelectors: {
      "Audio Selector 1": {
        Offset: 0,
        DefaultSelection: "NOT_DEFAULT",
        ProgramSelection: 1,
        SelectorType: "TRACK",
        Tracks: [1],
      },
    },
    VideoSelector: {
      ColorSpace: "FOLLOW",
    },
    FilterEnable: "AUTO",
    PsiControl: "USE_PSI",
    FilterStrength: 0,
    DeblockFilter: "DISABLED",
    DenoiseFilter: "DISABLED",
    TimecodeSource: "EMBEDDED",
  },
],
TimecodeConfig: {
  Source: "EMBEDDED",
},
},
};
```

```
const run = async () => {
  try {
    // Create a promise on a MediaConvert object
    const data = await emcClient.send(new CreateJobTemplateCommand(params));
    console.log("Success!", data);
    return data;
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node emc_create_jobtemplate.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Einen Transcodierungsauftrag aus einer Jobvorlage erstellen

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `emc_template_createjob.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor gezeigt konfigurieren, einschließlich der Installation der erforderlichen Clients und Pakete.

Erstellen Sie die Auftragserstellungsparameter-JSON, einschließlich dem Namen der zu verwendenden Auftragsvorlage und der zu verwendenden Settings, die spezifisch für den Auftrag sind, den Sie erstellen. Rufen Sie dann die `CreateJobsCommand` Methode auf, indem Sie ein Versprechen zum Aufrufen eines MediaConvert Client-Dienstobjekts erstellen und die Parameter übergeben.

Note

JOB_QUEUE_ARN Ersetzen Sie es durch den Amazon-Ressourcennamen (ARN) der zu prüfenden Job-Warteschlange, durch, ***TEMPLATE_NAME ROLE_ARN*** mit, durch den Amazon-Ressourcennamen (ARN) der Rolle sowie ***INPUT_BUCKET_AND_FILENAME*** durch den Eingabe-Bucket und den Dateinamen — zum Beispiel `"s3://BUCKET_NAME/FILE_NAME"`.
KEY_PAIR_NAME

```
// Import required AWS-SDK clients and commands for Node.js
import { CreateJobCommand } from "@aws-sdk/client-mediaconvert";
import { emcClient } from "../libs/emcClient.js";

const params = {
  Queue: "QUEUE_ARN", //QUEUE_ARN
  JobTemplate: "TEMPLATE_NAME", //TEMPLATE_NAME
  Role: "ROLE_ARN", //ROLE_ARN
  Settings: {
    Inputs: [
      {
        AudioSelectors: {
          "Audio Selector 1": {
            Offset: 0,
            DefaultSelection: "NOT_DEFAULT",
            ProgramSelection: 1,
            SelectorType: "TRACK",
            Tracks: [1],
          },
        },
        VideoSelector: {
          ColorSpace: "FOLLOW",
        },
        FilterEnable: "AUTO",
        PsiControl: "USE_PSI",
        FilterStrength: 0,
        DeblockFilter: "DISABLED",
        DenoiseFilter: "DISABLED",
        TimecodeSource: "EMBEDDED",
        FileInput: "INPUT_BUCKET_AND_FILENAME", //INPUT_BUCKET_AND_FILENAME, e.g.,
        "s3://BUCKET_NAME/FILE_NAME"
      },
    ],
  },
};

const run = async () => {
  try {
    const data = await emcClient.send(new CreateJobCommand(params));
    console.log("Success! ", data);
    return data;
  } catch (err) {
    console.log("Error", err);
  }
};
```

```
}  
};  
run();
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node emc_template_createjob.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Listet Ihre Jobvorlagen auf

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `emc_listtemplates.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor gezeigt konfigurieren, einschließlich der Installation der erforderlichen Clients und Pakete.

Erstellen Sie ein Objekt, um die Anfrageparameter für die `listTemplates`-Methode der `MediaConvert`-Client-Klasse zu übergeben. Schließen Sie Werte ein, um zu bestimmen, welche Vorlagen gelistet werden sollen (`NAME`, `CREATION DATE`, `SYSTEM`), wie viele und deren Sortierreihenfolge. Um die `ListTemplatesCommand` Methode aufzurufen, erstellen Sie ein Versprechen zum Aufrufen eines `MediaConvert` Client-Dienstobjekts und übergeben dabei die Parameter.

```
// Import required AWS-SDK clients and commands for Node.js  
import { ListJobTemplatesCommand } from "@aws-sdk/client-mediaconvert";  
import { emcClient } from "../libs/emcClient.js";  
  
const params = {  
  ListBy: "NAME",  
  MaxResults: 10,  
  Order: "ASCENDING",  
};  
  
const run = async () => {  
  try {  
    const data = await emcClient.send(new ListJobTemplatesCommand(params));  
    console.log("Success ", data.JobTemplates);  
    return data;  
  } catch (err) {  
    console.log("Error", err);  
  }  
}
```

```
}  
};  
run();
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node emc_listtemplates.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Löschen einer Jobvorlage

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `emc_deletetemplate.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor gezeigt konfigurieren, einschließlich der Installation der erforderlichen Clients und Pakete.

Erstellen Sie ein Objekt, um den Namen der Auftragsvorlage, die Sie löschen möchten, als Parameter für die `DeleteJobTemplateCommand`-Methode der `MediaConvert`-Client-Klasse zu übergeben. Um die `DeleteJobTemplateCommand` Methode aufzurufen, erstellen Sie ein Versprechen zum Aufrufen eines `MediaConvert` Client-Dienstobjekts und übergeben dabei die Parameter.

```
// Import required AWS-SDK clients and commands for Node.js  
import { DeleteJobTemplateCommand } from "@aws-sdk/client-mediaconvert";  
import { emcClient } from "../libs/emcClient.js";  
  
// Set the parameters  
const params = { Name: "test" }; //TEMPLATE_NAME  
  
const run = async () => {  
  try {  
    const data = await emcClient.send(new DeleteJobTemplateCommand(params));  
    console.log(  
      "Success, template deleted! Request ID:",  
      data.$metadata.requestId,  
    );  
    return data;  
  } catch (err) {  
    console.log("Error", err);  
  }  
};  
run();
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node emc_deletetemplate.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

AWS Lambda Beispiele

AWS Lambda ist ein serverloser Rechendienst, mit dem Sie Code ausführen können, ohne Server bereitzustellen oder zu verwalten, eine auslastungsabhängige Cluster-Skalierungslogik zu erstellen, Eventintegrationen zu verwalten oder Laufzeiten zu verwalten.

Die JavaScript API für wird über die Client-Klasse verfügbar AWS Lambda gemacht. [LambdaService](#)

Hier ist eine Liste von Beispielen, die zeigen, wie Lambda-Funktionen mit der AWS SDK für JavaScript Version 3 erstellt und verwendet werden:

- [Lambda mit API Gateway aufrufen](#)
- [Geplante Ereignisse zur Ausführung von AWS Lambda Funktionen erstellen](#)

Amazon-Lex-Beispiele

Amazon Lex ist ein AWS Service zum Integrieren von Konversationsschnittstellen in Anwendungen, die Sprache und Text verwenden.

Die JavaScript API für Amazon Lex wird über die [Lex Runtime Service-Clientklasse](#) verfügbar gemacht.

- [Erstellen eines Amazon-Lex-Chatbots](#)

Beispiele für Amazon Polly



Dieses Node.js-Codebeispiel zeigt:

- Laden Sie mit Amazon Polly aufgenommene Audiodaten auf Amazon S3 hoch

Das Szenario

In diesem Beispiel werden eine Reihe von Node.js -Modulen verwendet, um mit Amazon Polly aufgezeichnetes Audio mithilfe der folgenden Methoden der Amazon S3-Clientklasse automatisch auf Amazon S3 hochzuladen:

- [StartSpeechSynthesisTaskCommand](#)

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Richten Sie eine Projektumgebung ein, um JavaScript Node-Beispiele auszuführen, indem Sie den Anweisungen unter folgen [GitHub](#).
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zur Bereitstellung einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter [Dateien mit gemeinsam genutzten Konfigurationen und Anmeldeinformationen](#) im AWS SDKs Referenzhandbuch zu Tools.
- Erstellen Sie eine AWS Identity and Access Management (IAM) unauthentifizierte Amazon Cognito Cognito-Benutzerrolle polly: SynthesizeSpeech permissions und einen Amazon Cognito Cognito-Identitätspool mit der zugehörigen IAM-Rolle. Im folgenden [Erstellen Sie die Ressourcen mit dem AWS CloudFormation](#) Abschnitt wird beschrieben, wie Sie diese Ressourcen erstellen.

Note

In diesem Beispiel wird Amazon Cognito verwendet, aber wenn Sie Amazon Cognito nicht verwenden, muss Ihr AWS Benutzer über die folgenden IAM-Berechtigungsrichtlinien verfügen

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```

```
"Action": [
  "mobileanalytics:PutEvents",
  "cognito-sync:*"
],
"Resource": "*",
"Effect": "Allow"
},
{
  "Action": "polly:SynthesizeSpeech",
  "Resource": "*",
  "Effect": "Allow"
}
]
```

Erstellen Sie die Ressourcen mit dem AWS CloudFormation

CloudFormation ermöglicht es Ihnen, AWS Infrastrukturbereitstellungen vorhersehbar und wiederholt zu erstellen und bereitzustellen. [Weitere Informationen zu CloudFormation finden Sie im AWS CloudFormation Benutzerhandbuch.](#)

Um den CloudFormation Stapel zu erstellen:

1. Installieren und konfigurieren Sie die AWS CLI folgenden Anweisungen im [AWS CLI Benutzerhandbuch](#).
2. Erstellen Sie eine Datei mit dem Namen `setup.yaml` im Stammverzeichnis Ihres Projektordners und kopieren Sie den Inhalt [hier GitHub](#) hinein.

Note

Die CloudFormation Vorlage wurde unter Verwendung der [hier AWS CDK verfügbaren Datei](#) generiert GitHub. Weitere Informationen zu finden Sie im [AWS Cloud Development Kit \(AWS CDK\) Entwicklerhandbuch](#). AWS CDK

3. Führen Sie den folgenden Befehl von der Befehlszeile aus und **STACK_NAME** ersetzen Sie ihn durch einen eindeutigen Namen für den Stack.

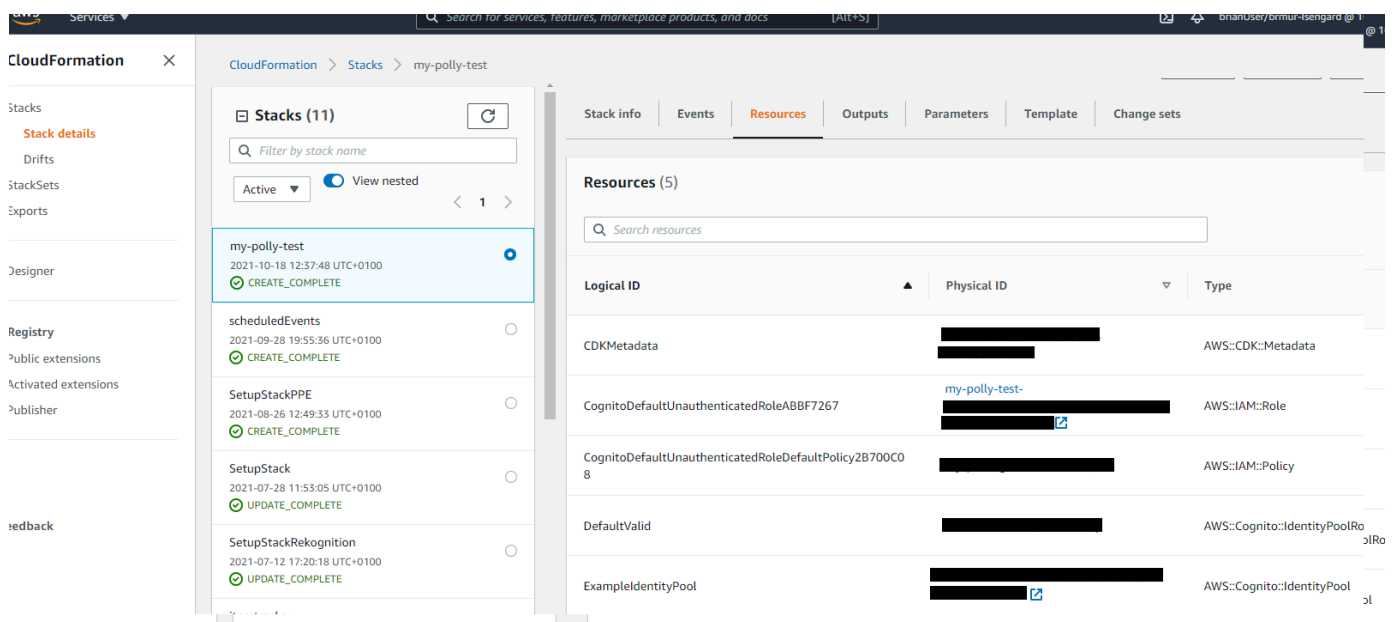
⚠ Important

Der Stack-Name muss innerhalb einer AWS Region und eines AWS Kontos eindeutig sein. Sie können bis zu 128 Zeichen angeben. Zahlen und Bindestriche sind zulässig.

```
aws cloudformation create-stack --stack-name STACK_NAME --template-body file://
setup.yaml --capabilities CAPABILITY_IAM
```

Weitere Informationen zu den create-stack Befehlsparametern finden Sie in der [AWS CLI Befehlsreferenz](#) und im [CloudFormation Benutzerhandbuch](#).

4. Navigieren Sie zur CloudFormation Managementkonsole, wählen Sie Stacks, wählen Sie den Stack-Namen und klicken Sie auf die Registerkarte Ressourcen, um eine Liste der erstellten Ressourcen anzuzeigen.



Laden Sie mit Amazon Polly aufgenommene Audiodaten auf Amazon S3 hoch

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `polly_synthesize_to_s3.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor gezeigt konfigurieren, einschließlich der Installation der erforderlichen Clients und Pakete. Geben Sie im Code die **REGION**, und die **BUCKET_NAME**. Um auf Amazon Polly zuzugreifen, erstellen Sie ein Polly Kundenservice-Objekt. **"IDENTITY_POOL_ID"** Ersetzen Sie es IdentityPoolId durch die Seite „Beispiel“ des Amazon

Cognito Cognito-Identitätspools, den Sie für dieses Beispiel erstellt haben. Dies wird auch an jedes Client-Objekt übergeben.

Rufen Sie die `StartSpeechSynthesisCommand` Methode des Amazon Polly Polly-Client-Serviceobjekts auf, synthetisieren Sie die Sprachnachricht und laden Sie sie in den Amazon S3 S3-Bucket hoch.

```
import { StartSpeechSynthesisTaskCommand } from "@aws-sdk/client-polly";
import { pollyClient } from "../libs/pollyClient.js";

// Create the parameters
const params = {
  OutputFormat: "mp3",
  OutputS3BucketName: "videoanalyzerbucket",
  Text: "Hello David, How are you?",
  TextType: "text",
  VoiceId: "Joanna",
  SampleRate: "22050",
};

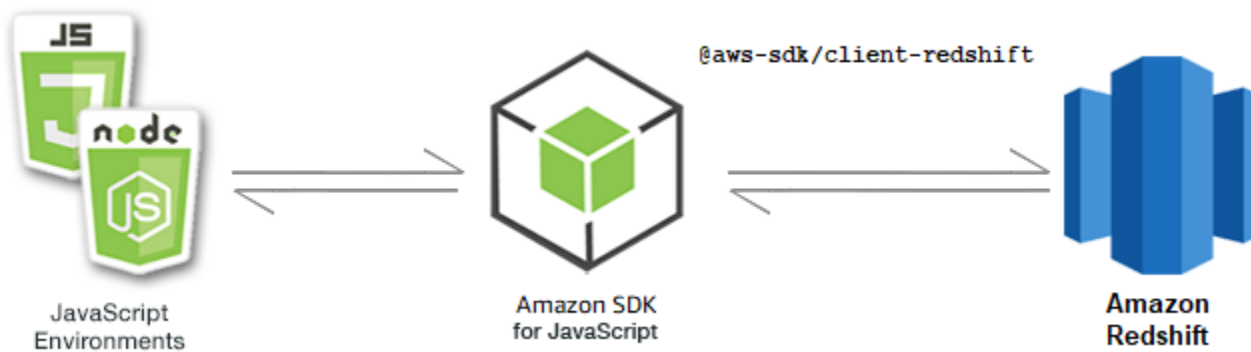
const run = async () => {
  try {
    await pollyClient.send(new StartSpeechSynthesisTaskCommand(params));
    console.log(`Success, audio file added to ${params.OutputS3BucketName}`);
  } catch (err) {
    console.log("Error putting object", err);
  }
};

run();
```

Diesen Beispielcode finden Sie [hier auf GitHub](#)

Beispiele für Amazon Redshift

Amazon Redshift ist ein vollständig verwalteter Data-Warehouse-Service in Petabytegröße in der Cloud. Ein Amazon-Redshift-Data-Warehouse ist eine Sammlung von Datenverarbeitungsressourcen, den so genannten Knoten, die zu Gruppen, den so genannten Clustern, zusammengefasst werden. In jedem Cluster wird eine Amazon-Redshift-Engine ausgeführt, und er enthält mindestens eine Datenbank.



Die JavaScript API für Amazon Redshift wird über die [Amazon Redshift Redshift-Client-Klasse](#) verfügbar gemacht.

Themen

- [Beispiele für Amazon Redshift](#)

Beispiele für Amazon Redshift

In diesem Beispiel werden eine Reihe von Node.js -Modulen verwendet, um Amazon Redshift Redshift-Cluster mit den folgenden Methoden der Redshift Client-Klasse zu erstellen, zu ändern, zu beschreiben und anschließend zu löschen:

- [CreateClusterCommand](#)
- [ModifyClusterCommand](#)
- [DescribeClustersCommand](#)
- [DeleteClusterCommand](#)

Weitere Informationen zu Amazon Redshift-Benutzern finden Sie im [Amazon Redshift Getting Started Guide](#).

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Richten Sie die Projektumgebung ein, um diese TypeScript Node-Beispiele auszuführen, und installieren Sie die erforderlichen Module AWS SDK für JavaScript und Module von Drittanbietern. Folgen Sie den Anweisungen auf [GitHub](#).

- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zur Bereitstellung einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter [Dateien mit gemeinsam genutzten Konfigurationen und Anmeldeinformationen](#) im AWS SDKs Referenzhandbuch zu Tools.

 Important

Diese Beispiele zeigen, wie Client-Dienstobjekte und Befehle mithilfe von ECMAScript6 (ES6) importiert/exportiert werden.

- Dies erfordert Node.js Version 13.x oder höher. Informationen zum Herunterladen und Installieren der neuesten Version von Node.js finden Sie unter [Node.js downloads](#).
- Wenn Sie die CommonJS-Syntax bevorzugen, finden Sie weitere Informationen unter [JavaScript ES6/CommonJs-Syntax](#)

Einen Amazon Redshift Redshift-Cluster erstellen

Dieses Beispiel zeigt, wie Sie mit dem einen Amazon Redshift Redshift-Cluster erstellen. AWS SDK für JavaScript Weitere Informationen finden Sie unter [CreateCluster](#).

 Important

Der Cluster, den Sie erstellen möchten, ist live (und läuft nicht in einer Sandbox). Es fallen so lange die standardmäßigen Amazon-Redshift-Nutzungsgebühren für den Cluster an, bis Sie ihn löschen. Wenn Sie den Cluster in derselben Sitzung löschen, in der Sie ihn erstellt haben, sind die Gesamtkosten minimal.

Erstellen Sie ein `libs` Verzeichnis und ein Modul `Node.js` mit dem Dateinamen `redshiftClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon Redshift Redshift-Client-Objekt erstellt wird. Ersetzen Sie es **REGION** durch Ihre AWS Region.

```
import { RedshiftClient } from "@aws-sdk/client-redshift";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create Redshift service object.
```

```
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `redshift-create-cluster.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor gezeigt konfigurieren, einschließlich der Installation der erforderlichen Clients und Pakete. Erstellen Sie ein Parameterobjekt, das den Knotentyp, der bereitgestellt werden soll, und die Master-Anmeldedaten für die automatisch im Cluster erstellte Datenbankinstanz und schließlich den Clustertyp angibt.

Note

Ersetzen Sie es `CLUSTER_NAME` durch den Namen des Clusters. `NODE_TYPE` Geben Sie zum Beispiel den Knotentyp an, der bereitgestellt werden soll, z. B. „dc2.large“. `MASTER_USERNAME` und `MASTER_USER_PASSWORD` sind die Anmeldeinformationen des Masterbenutzers Ihrer DB-Instance im Cluster. Geben Sie für `CLUSTER_TYPE` den Clustertyp ein. Wenn Sie `single-node` angeben, benötigen Sie den `NumberOfNodes` Parameter nicht. Die übrigen Parameter sind optional.

```
// Import required AWS SDK clients and commands for Node.js
import { CreateClusterCommand } from "@aws-sdk/client-redshift";
import { redshiftClient } from "../libs/redshiftClient.js";

const params = {
  ClusterIdentifier: "CLUSTER_NAME", // Required
  NodeType: "NODE_TYPE", //Required
  MasterUsername: "MASTER_USER_NAME", // Required - must be lowercase
  MasterUserPassword: "MASTER_USER_PASSWORD", // Required - must contain at least one
  uppercase letter, and one number
  ClusterType: "CLUSTER_TYPE", // Required
  IAMRoleARN: "IAM_ROLE_ARN", // Optional - the ARN of an IAM role with permissions
  your cluster needs to access other AWS services on your behalf, such as Amazon S3.
  ClusterSubnetGroupName: "CLUSTER_SUBNET_GROUPNAME", //Optional - the name of a
  cluster subnet group to be associated with this cluster. Defaults to 'default' if not
  specified.
  DBName: "DATABASE_NAME", // Optional - defaults to 'dev' if not specified
  Port: "PORT_NUMBER", // Optional - defaults to '5439' if not specified
};
```

```
const run = async () => {
  try {
    const data = await redshiftClient.send(new CreateClusterCommand(params));
    console.log(
      `Cluster ${data.Cluster.ClusterIdentifier} successfully created`,
    );
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node redshift-create-cluster.js
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Ändern eines Amazon Redshift Redshift-Clusters

Dieses Beispiel zeigt, wie Sie das Master-Benutzerkennwort eines Amazon Redshift Redshift-Clusters mithilfe von `changeMasterUserPassword` ändern. AWS SDK für JavaScript Weitere Informationen darüber, welche anderen Einstellungen Sie ändern können, finden Sie unter [ModifyCluster](#).

Erstellen Sie ein `libs` Verzeichnis und ein Modul `Node.js` mit dem Dateinamen `redshiftClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon Redshift Redshift-Client-Objekt erstellt wird. Ersetzen Sie es **REGION** durch Ihre AWS Region.

```
import { RedshiftClient } from "@aws-sdk/client-redshift";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create Redshift service object.
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein `Node.js`-Modul mit dem Dateinamen `redshift-modify-cluster.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor gezeigt konfigurieren, einschließlich der Installation der

erforderlichen Clients und Pakete. Geben Sie die AWS Region, den Namen des Clusters, den Sie ändern möchten, und das neue Masterbenutzerkennwort an.

 Note

CLUSTER_NAME Ersetzen Sie es durch den Namen des Clusters und *MASTER_USER_PASSWORD* durch das neue Masterbenutzerkennwort.

```
// Import required AWS SDK clients and commands for Node.js
import { ModifyClusterCommand } from "@aws-sdk/client-redshift";
import { redshiftClient } from "../libs/redshiftClient.js";

// Set the parameters
const params = {
  ClusterIdentifier: "CLUSTER_NAME",
  MasterUserPassword: "NEW_MASTER_USER_PASSWORD",
};

const run = async () => {
  try {
    const data = await redshiftClient.send(new ModifyClusterCommand(params));
    console.log("Success was modified.", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node redshift-modify-cluster.js
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Details eines Amazon Redshift Redshift-Clusters anzeigen

Dieses Beispiel zeigt, wie Sie die Details eines Amazon Redshift Redshift-Clusters mithilfe von anzeigen. AWS SDK für JavaScript Weitere Informationen zu optionalen Optionen finden Sie unter [DescribeClusters](#).

Erstellen Sie ein `libs` Verzeichnis und ein Modul `Node.js` mit dem Dateinamen `redshiftClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon Redshift Redshift-Client-Objekt erstellt wird. Ersetzen Sie es **REGION** durch Ihre AWS Region.

```
import { RedshiftClient } from "@aws-sdk/client-redshift";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create Redshift service object.
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein `Node.js`-Modul mit dem Dateinamen `redshift-describe-clusters.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor gezeigt konfigurieren, einschließlich der Installation der erforderlichen Clients und Pakete. Geben Sie die AWS Region, den Namen des Clusters, den Sie ändern möchten, und das neue Masterbenutzerkennwort an.

 Note

CLUSTER_NAME Ersetzen Sie es durch den Namen des Clusters.

```
// Import required AWS SDK clients and commands for Node.js
import { DescribeClustersCommand } from "@aws-sdk/client-redshift";
import { redshiftClient } from "../libs/redshiftClient.js";

const params = {
  ClusterIdentifier: "CLUSTER_NAME",
};

const run = async () => {
  try {
    const data = await redshiftClient.send(new DescribeClustersCommand(params));
    console.log("Success", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
```

```
run();
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node redshift-describe-clusters.js
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Löschen Sie einen Amazon Redshift Redshift-Cluster

Dieses Beispiel zeigt, wie Sie die Details eines Amazon Redshift Redshift-Clusters mithilfe von anzeigen. AWS SDK für JavaScript Weitere Informationen darüber, welche anderen Einstellungen Sie ändern können, finden Sie unter [DeleteCluster](#).

Erstellen Sie ein `libs` Verzeichnis und ein Modul `Node.js` mit dem Dateinamen `redshiftClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon Redshift Redshift-Client-Objekt erstellt wird. Ersetzen Sie es **REGION** durch Ihre AWS Region.

```
import { RedshiftClient } from "@aws-sdk/client-redshift";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create Redshift service object.
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Modul `Node.js` mit dem Namen der Datei `redshift-delete-clusters.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor gezeigt konfigurieren, einschließlich der Installation der erforderlichen Clients und Pakete. Geben Sie die AWS Region, den Namen des Clusters, den Sie ändern möchten, und das neue Masterbenutzerkennwort an. Geben Sie dann an, ob Sie vor dem Löschen einen endgültigen Snapshot des Clusters speichern möchten, und wenn ja, die ID des Snapshots.

Note

CLUSTER_NAME Ersetzen Sie durch den Namen des Clusters. Geben Sie für den Parameter **SkipFinalClusterSnapshot**, ob vor dem Löschen ein letzter Snapshot des Clusters erstellt werden soll. Wenn Sie 'false' angeben, geben Sie die ID des endgültigen Cluster-

Snapshots in an **CLUSTER_SNAPSHOT_ID**. Sie können diese ID abrufen, indem Sie im Cluster-Dashboard auf den Link in der Spalte Snapshots für den Cluster klicken und nach unten zum Bereich Snapshots scrollen. Beachten Sie, dass der Stamm nicht Teil der Snapshot-ID `rs:` ist.

```
// Import required AWS SDK clients and commands for Node.js
import { DeleteClusterCommand } from "@aws-sdk/client-redshift";
import { redshiftClient } from "../libs/redshiftClient.js";

const params = {
  ClusterIdentifier: "CLUSTER_NAME",
  SkipFinalClusterSnapshot: false,
  FinalClusterSnapshotIdentifier: "CLUSTER_SNAPSHOT_ID",
};

const run = async () => {
  try {
    const data = await redshiftClient.send(new DeleteClusterCommand(params));
    console.log("Success, cluster deleted. ", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};

run();
```

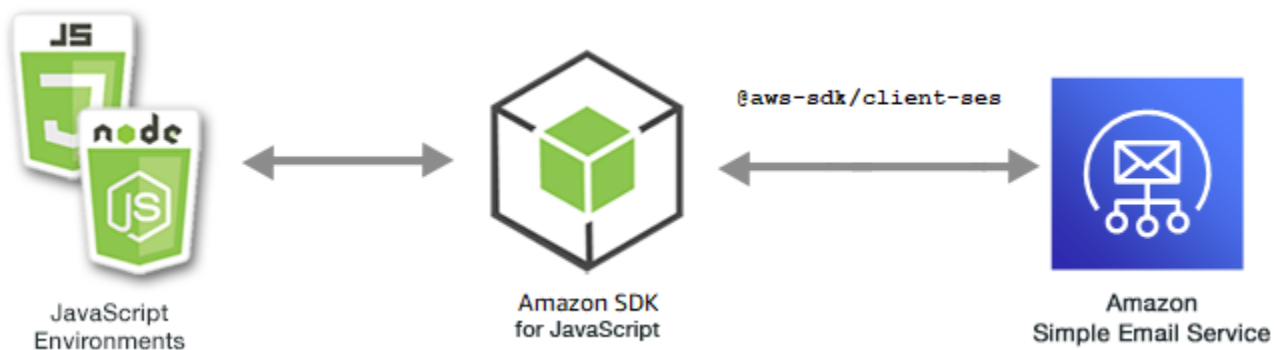
Um das Beispiel auszuführen, geben Sie in der Befehlszeile Folgendes ein.

```
node redshift-delete-cluster.js
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Beispiele für Amazon Simple Email Service

Amazon Simple Email Service (Amazon SES) ist ein cloudbasierter E-Mail-Versandservice, der digitale Vermarkter und Anwendungsentwickler beim Versenden von Marketing-, Benachrichtigungs- und Transaktions-E-Mails unterstützt. Dabei handelt es sich um einen zuverlässigen, kosteneffektiven Dienst für Unternehmen jeder Größe, die den Kontakt zu ihren Kunden mithilfe von E-Mail aufrechterhalten



Die JavaScript API für Amazon SES wird über die SES Client-Klasse verfügbar gemacht. Weitere Informationen zur Verwendung der Amazon SES SES-Client-Klasse finden Sie unter [Class: SES](#) in der API-Referenz.

Themen

- [Verwaltung von Amazon SES SES-Identitäten](#)
- [Arbeiten mit E-Mail-Vorlagen in Amazon SES](#)
- [Senden von E-Mails mit Amazon SES](#)

Verwaltung von Amazon SES SES-Identitäten



Dieses Node.js-Codebeispiel zeigt:

- So verifizieren Sie E-Mail-Adressen und Domains, die mit Amazon SES verwendet werden.
- So weisen Sie Ihren Amazon SES SES-Identitäten eine AWS Identity and Access Management (IAM-) Richtlinie zu.
- So listen Sie alle Amazon SES SES-Identitäten für Ihr AWS Konto auf.
- So löschen Sie Identitäten, die mit Amazon SES verwendet werden.

Eine Amazon SES-Identität ist eine E-Mail-Adresse oder Domain, die Amazon SES zum Senden von E-Mails verwendet. Amazon SES verlangt von Ihnen, Ihre E-Mail-Identitäten zu verifizieren, um zu bestätigen, dass sie Ihnen gehören, und zu verhindern, dass andere sie verwenden.

Einzelheiten zur Verifizierung von E-Mail-Adressen und Domains in Amazon SES finden Sie unter [Verifizieren von E-Mail-Adressen und Domains in Amazon SES](#) im Amazon Simple Email Service Developer Guide. Informationen zur Sendeautorisierung in Amazon SES finden Sie unter [Überblick über die Amazon SES SES-Sendeautorisierung](#).

Das Szenario

In diesem Beispiel verwenden Sie eine Reihe von Node.js -Modulen, um Amazon SES SES-Identitäten zu überprüfen und zu verwalten. Die Module Node.js verwenden das SDK JavaScript zur Überprüfung von E-Mail-Adressen und Domains und verwenden dabei die folgenden Methoden der SES Client-Klasse:

- [ListIdentitiesCommand](#)
- [DeleteIdentityCommand](#)
- [VerifyEmailIdentityCommand](#)
- [VerifyDomainIdentityCommand](#)

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Richten Sie die Projektumgebung ein, um diese TypeScript Node-Beispiele auszuführen, und installieren Sie die erforderlichen Module AWS SDK für JavaScript und Module von Drittanbietern. Folgen Sie den Anweisungen auf [GitHub](#).
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zur Bereitstellung einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter [Dateien mit gemeinsam genutzten Konfigurationen und Anmeldeinformationen](#) im AWS SDKs Referenzhandbuch zu Tools.

Important

Diese Beispiele zeigen, wie Client-Dienstobjekte und Befehle mithilfe von ECMAScript6 (ES6) importiert/exportiert werden.

- Dazu ist die Version 13.x von Node.js oder höher erforderlich. Informationen zum Herunterladen und Installieren der neuesten Version von Node.js finden Sie unter [Node.js downloads](#).

- Wenn Sie die CommonJS-Syntax bevorzugen, finden Sie weitere Informationen unter [JavaScript ES6/CommonJs-Syntax](#).

Auflisten Ihrer Identitäten

Verwenden Sie in diesem Beispiel ein Modul Node.js, um E-Mail-Adressen und Domains aufzulisten, die mit Amazon SES verwendet werden sollen.

Erstellen Sie ein `libs` Verzeichnis und ein Modul Node.js mit dem Dateinamen `sesClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SES SES-Client-Objekt erstellt wird. **REGION** Ersetzen Sie es durch Ihre AWS Region.

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ses_listidentities.js`. Konfigurieren Sie das SDK wie zuvor gezeigt, einschließlich der Installation der erforderlichen Clients und Pakete.

Erstellen Sie ein Objekt, um den `IdentityType` und andere Parameter für die `ListIdentitiesCommand`-Methode der SES-Client-Klasse zu übergeben. Um die `ListIdentitiesCommand` Methode aufzurufen, rufen Sie ein Amazon SES SES-Serviceobjekt auf und übergeben das Parameter-Objekt.

Das data zurückgegebene Objekt enthält ein Array von Domänenidentitäten, wie durch den `IdentityType` Parameter angegeben.

Note

IdentityType Ersetzen Sie es durch den Identitätstyp, der "EmailAddress" oder „Domain“ sein kann.

```
import { ListIdentitiesCommand } from "@aws-sdk/client-ses";
```

```
import { sesClient } from "../libs/sesClient.js";

const createListIdentitiesCommand = () =>
  new ListIdentitiesCommand({ IdentityType: "EmailAddress", MaxItems: 10 });

const run = async () => {
  const listIdentitiesCommand = createListIdentitiesCommand();

  try {
    return await sesClient.send(listIdentitiesCommand);
  } catch (err) {
    console.log("Failed to list identities.", err);
    return err;
  }
};
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node ses_listidentities.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Verifizieren der Identität einer E-Mail-Adresse

Verwenden Sie in diesem Beispiel ein Modul Node.js, um E-Mail-Absender für die Verwendung mit Amazon SES zu verifizieren.

Erstellen Sie ein `libs` Verzeichnis und ein Modul Node.js mit dem Dateinamens `sesClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SES SES-Client-Objekt erstellt wird. **REGION** Ersetzen Sie es durch Ihre AWS Region.

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ses_verifyemailidentity.js`. Konfigurieren Sie das SDK wie zuvor gezeigt, einschließlich des Herunterladens der erforderlichen Clients und Pakete.

Erstellen Sie ein Objekt mit dem sie den `EmailAddress`-Parameter an die `VerifyEmailIdentityCommand`-Methode der `SES-Client`-Klasse übergeben. Um die `VerifyEmailIdentityCommand` Methode aufzurufen, rufen Sie ein Amazon SES `SES-Client`-Serviceobjekt auf und übergeben die Parameter.

 Note

EMAIL_ADDRESS Ersetzen Sie es durch die E-Mail-Adresse, z. B. `name@example.com`.

```
// Import required AWS SDK clients and commands for Node.js
import { VerifyEmailIdentityCommand } from "@aws-sdk/client-ses";
import { sesClient } from "../libs/sesClient.js";

const EMAIL_ADDRESS = "name@example.com";

const createVerifyEmailIdentityCommand = (emailAddress) => {
  return new VerifyEmailIdentityCommand({ EmailAddress: emailAddress });
};

const run = async () => {
  const verifyEmailIdentityCommand =
    createVerifyEmailIdentityCommand(EMAIL_ADDRESS);
  try {
    return await sesClient.send(verifyEmailIdentityCommand);
  } catch (err) {
    console.log("Failed to verify email identity.", err);
    return err;
  }
};
```

Um das Beispiel auszuführen, geben Sie in der Befehlszeile Folgendes ein. Die Domain wird zu Amazon SES hinzugefügt, um verifiziert zu werden.

```
node ses_verifyemailidentity.js
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Überprüfung einer Domänenidentität

Verwenden Sie in diesem Beispiel ein Modul Node.js, um E-Mail-Domänen für die Verwendung mit Amazon SES zu verifizieren.

Erstellen Sie ein `libs` Verzeichnis und ein Modul Node.js mit dem Dateinamen `sesClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SES SES-Client-Objekt erstellt wird. **REGION** Ersetzen Sie es durch Ihre AWS Region.

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ses_verifydomainidentity.js`. Konfigurieren Sie das SDK wie zuvor gezeigt, einschließlich der Installation der erforderlichen Clients und Pakete.

Erstellen Sie ein Objekt mit dem sie den Domain-Parameter an die `VerifyDomainIdentityCommand`-Methode der SES-Client-Klasse übergeben. Um die `VerifyDomainIdentityCommand` Methode aufzurufen, rufen Sie ein Amazon SES SES-Client-Serviceobjekt auf und übergeben das Parameter-Objekt.

Note

In diesem Beispiel werden die erforderlichen AWS Service V3-Paketclients und V3-Befehle importiert und verwendet und die `send` Methode in einem Async/Await-Muster verwendet. Sie können dieses Beispiel stattdessen mit V2-Befehlen erstellen, indem Sie einige geringfügige Änderungen vornehmen. Details hierzu finden Sie unter [Verwenden von v3-Befehlen](#).

Note

DOMAIN_NAME Ersetzen Sie es durch den Domainnamen.

```
import { VerifyDomainIdentityCommand } from "@aws-sdk/client-ses";
import {
  getUniqueName,
  postfix,
} from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "../libs/sesClient.js";

/**
 * You must have access to the domain's DNS settings to complete the
 * domain verification process.
 */
const DOMAIN_NAME = postfix(getUniqueName("Domain"), ".example.com");

const createVerifyDomainIdentityCommand = () => {
  return new VerifyDomainIdentityCommand({ Domain: DOMAIN_NAME });
};

const run = async () => {
  const VerifyDomainIdentityCommand = createVerifyDomainIdentityCommand();

  try {
    return await sesClient.send(VerifyDomainIdentityCommand);
  } catch (err) {
    console.log("Failed to verify domain.", err);
    return err;
  }
};
```

Um das Beispiel auszuführen, geben Sie in der Befehlszeile Folgendes ein. Die Domain wird zu Amazon SES hinzugefügt, um verifiziert zu werden.

```
node ses_verifydomainidentity.js
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Identitäten löschen

Verwenden Sie in diesem Beispiel ein Modul Node.js, um E-Mail-Adressen oder Domains zu löschen, die mit Amazon SES verwendet werden.

Erstellen Sie ein `libs` Verzeichnis und ein Modul `Node.js` mit dem Dateinamen `sesClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SES `SES-Client`-Objekt erstellt wird. **REGION** Ersetzen Sie es durch Ihre AWS Region.

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein `Node.js`-Modul mit dem Dateinamen `ses_deleteidentity.js`. Konfigurieren Sie das SDK wie zuvor gezeigt, einschließlich der Installation der erforderlichen Clients und Pakete.

Erstellen Sie ein Objekt mit dem sie den `Identity`-Parameter an die `DeleteIdentityCommand`-Methode der `SES-Client`-Klasse übergeben. Um die `DeleteIdentityCommand` Methode aufzurufen, erstellen Sie ein Serviceobjekt `request` für den Aufruf eines Amazon SES `SES-Clients` und übergeben dabei die Parameter.

Note

In diesem Beispiel werden die erforderlichen AWS Service V3-Paketclients und V3-Befehle importiert und verwendet und die `send` Methode in einem `Async/Await`-Muster verwendet. Sie können dieses Beispiel stattdessen mit V2-Befehlen erstellen, indem Sie einige geringfügige Änderungen vornehmen. Details hierzu finden Sie unter [Verwenden von v3-Befehlen](#).

Note

IDENTITY_EMAIL Ersetzen Sie es durch die E-Mail-Adresse der Identität, die gelöscht werden soll.

```
import { DeleteIdentityCommand } from "@aws-sdk/client-ses";
import { sesClient } from "../libs/sesClient.js";

const IDENTITY_EMAIL = "fake@example.com";
```

```
const createDeleteIdentityCommand = (identityName) => {
  return new DeleteIdentityCommand({
    Identity: identityName,
  });
};

const run = async () => {
  const deleteIdentityCommand = createDeleteIdentityCommand(IDENTITY_EMAIL);

  try {
    return await sesClient.send(deleteIdentityCommand);
  } catch (err) {
    console.log("Failed to delete identity.", err);
    return err;
  }
};
```

Um das Beispiel auszuführen, geben Sie in der Befehlszeile Folgendes ein.

```
node ses_deleteidentity.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Arbeiten mit E-Mail-Vorlagen in Amazon SES



Dieses Node.js-Codebeispiel zeigt:

- So erhalten Sie eine Liste all Ihrer E-Mail-Vorlagen.
- So rufen Sie E-Mail-Vorlagen ab und aktualisieren sie.
- So erstellen und löschen Sie E-Mail-Vorlagen.

Amazon SES ermöglicht es Ihnen, personalisierte E-Mail-Nachrichten mithilfe von E-Mail-Vorlagen zu versenden. Einzelheiten zum Erstellen und Verwenden von E-Mail-Vorlagen in Amazon SES finden Sie unter [Senden personalisierter E-Mails mit der Amazon SES SES-API](#) im Amazon Simple Email Service Developer Guide.

Das Szenario

In diesem Beispiel verwenden Sie eine Reihe von Node.js-Module, um mit E-Mail-Vorlagen zu arbeiten. Die Module von Node.js verwenden das SDK JavaScript, um E-Mail-Vorlagen mithilfe der folgenden Methoden der SES Client-Klasse zu erstellen und zu verwenden:

- [ListTemplatesCommand](#)
- [CreateTemplateCommand](#)
- [GetTemplateCommand](#)
- [DeleteTemplateCommand](#)
- [UpdateTemplateCommand](#)

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Richten Sie die Projektumgebung ein, um diese TypeScript Node-Beispiele auszuführen, und installieren Sie die erforderlichen Module AWS SDK für JavaScript und Module von Drittanbietern. Folgen Sie den Anweisungen auf [GitHub](#).
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zur Bereitstellung einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter [Dateien mit gemeinsam genutzten Konfigurationen und Anmeldeinformationen](#) im AWS SDKs Referenzhandbuch zu Tools.

Important

Diese Beispiele zeigen, wie Client-Dienstobjekte und Befehle mithilfe von ECMAScript6 (ES6) importiert/exportiert werden.

- Dazu ist die Version 13.x von Node.js oder höher erforderlich. Informationen zum Herunterladen und Installieren der neuesten Version von Node.js finden Sie unter [Node.js downloads](#).
- Wenn Sie die CommonJS-Syntax bevorzugen, finden Sie weitere Informationen unter [JavaScript ES6/CommonJs-Syntax](#).

Ihre E-Mail-Vorlagen auflisten

Verwenden Sie in diesem Beispiel ein Modul Node.js, um eine E-Mail-Vorlage für Amazon SES zu erstellen.

Erstellen Sie ein `libs` Verzeichnis und ein Modul Node.js mit dem Dateinamen `sesClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SES SES-Client-Objekt erstellt wird. **REGION** Ersetzen Sie es durch Ihre AWS Region.

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ses_listtemplates.js`. Konfigurieren Sie das SDK wie zuvor gezeigt, einschließlich der Installation der erforderlichen Clients und Pakete.

Erstellen Sie ein Objekt, mit dem Sie die Parameter für die `ListTemplatesCommand`-Methode der SES-Client-Klasse übergeben können. Um die `ListTemplatesCommand` Methode aufzurufen, rufen Sie ein Amazon SES SES-Client-Serviceobjekt auf und übergeben die Parameter.

Note

In diesem Beispiel werden die erforderlichen AWS Service V3-Paketclients und V3-Befehle importiert und verwendet und die `send` Methode in einem Async/Await-Muster verwendet. Sie können dieses Beispiel stattdessen mit V2-Befehlen erstellen, indem Sie einige geringfügige Änderungen vornehmen. Details hierzu finden Sie unter [Verwenden von v3-Befehlen](#).

```
import { ListTemplatesCommand } from "@aws-sdk/client-ses";
import { sesClient } from "../libs/sesClient.js";

const createListTemplatesCommand = (maxItems) =>
  new ListTemplatesCommand({ MaxItems: maxItems });

const run = async () => {
```

```
const listTemplatesCommand = createListTemplatesCommand(10);

try {
  return await sesClient.send(listTemplatesCommand);
} catch (err) {
  console.log("Failed to list templates.", err);
  return err;
}
};
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein. Amazon SES gibt die Liste der Vorlagen zurück.

```
node ses_listtemplates.js
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Eine E-Mail-Vorlage erhalten

Verwenden Sie in diesem Beispiel ein Modul Node.js, um eine E-Mail-Vorlage für Amazon SES abzurufen.

Erstellen Sie ein `libs` Verzeichnis und ein Modul Node.js mit dem Dateinamen `sesClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SES SES-Client-Objekt erstellt wird. **REGION** Ersetzen Sie es durch Ihre AWS Region.

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ses_gettemplate.js`. Konfigurieren Sie das SDK wie zuvor gezeigt, einschließlich der Installation der erforderlichen Clients und Pakete.

Erstellen Sie ein Objekt mit dem sie den `TemplateName`-Parameter an die `GetTemplateCommand`-Methode der SES-Client-Klasse übergeben. Um die `GetTemplateCommand` Methode aufzurufen, rufen Sie ein Amazon SES SES-Client-Serviceobjekt auf und übergeben die Parameter.

Note

In diesem Beispiel werden die erforderlichen AWS Service V3-Paketclients und V3-Befehle importiert und verwendet und die `send` Methode in einem Async/Await-Muster verwendet. Sie können dieses Beispiel stattdessen mit V2-Befehlen erstellen, indem Sie einige geringfügige Änderungen vornehmen. Details hierzu finden Sie unter [Verwenden von v3-Befehlen](#).

Note

TEMPLATE_NAME Ersetzen Sie es durch den Namen der Vorlage, die zurückgegeben werden soll.

```
import { GetTemplateCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "../libs/sesClient.js";

const TEMPLATE_NAME = getUniqueName("TemplateName");

const createGetTemplateCommand = (templateName) =>
  new GetTemplateCommand({ TemplateName: templateName });

const run = async () => {
  const getTemplateCommand = createGetTemplateCommand(TEMPLATE_NAME);

  try {
    return await sesClient.send(getTemplateCommand);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected } */
      const messageRejectedError = caught;
      return messageRejectedError;
    }
    throw caught;
  }
};
```

Um das Beispiel auszuführen, geben Sie in der Befehlszeile Folgendes ein. Amazon SES gibt die Vorlagendetails zurück.

```
node ses_gettemplate.js
```

Dieser Beispielcode finden Sie [hier auf GitHub](#).

Eine E-Mail-Vorlage erstellen

Verwenden Sie in diesem Beispiel ein Modul Node.js, um eine E-Mail-Vorlage für Amazon SES zu erstellen.

Erstellen Sie ein `libs` Verzeichnis und ein Modul Node.js mit dem Dateinamen `sesClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SES SES-Client-Objekt erstellt wird. **REGION** Ersetzen Sie es durch Ihre AWS Region.

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ses_createtemplate.js`. Konfigurieren Sie das SDK wie zuvor gezeigt, einschließlich der Installation der erforderlichen Clients und Pakete.

Erstellen Sie ein Objekt, um die Parameter für die `CreateTemplateCommand`-Methode der SES-Client-Klasse zu übergeben, einschließlich `TemplateName`, `HtmlPart`, `SubjectPart` und `TextPart`. Um die `CreateTemplateCommand` Methode aufzurufen, rufen Sie ein Amazon SES SES-Client-Serviceobjekt auf und übergeben die Parameter.

Note

In diesem Beispiel werden die erforderlichen AWS Service V3-Paketclients und V3-Befehle importiert und verwendet und die `send` Methode in einem Async/Await-Muster verwendet. Sie können dieses Beispiel stattdessen mit V2-Befehlen erstellen, indem Sie einige geringfügige Änderungen vornehmen. Details hierzu finden Sie unter [Verwenden von v3-Befehlen](#).

 Note

TEMPLATE_NAME Ersetzen Sie es durch einen Namen für die neue Vorlage, **HtmlPart** durch den HTML-markierten Inhalt der E-Mail und **SubjectPart** durch den Betreff der E-Mail.

```
import { CreateTemplateCommand } from "@aws-sdk/client-ses";
import { sesClient } from "../libs/sesClient.js";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";

const TEMPLATE_NAME = getUniqueName("TestTemplateName");

const createCreateTemplateCommand = () => {
  return new CreateTemplateCommand({
    /**
     * The template feature in Amazon SES is based on the Handlebars template system.
     */
    Template: {
      /**
       * The name of an existing template in Amazon SES.
       */
      TemplateName: TEMPLATE_NAME,
      HtmlPart: `
        <h1>Hello, {{contact.firstName}}!</h1>
        <p>
          Did you know Amazon has a mascot named Peccy?
        </p>
      `,
      SubjectPart: "Amazon Tip",
    },
  });
};

const run = async () => {
  const createTemplateCommand = createCreateTemplateCommand();

  try {
    return await sesClient.send(createTemplateCommand);
  } catch (err) {
    console.log("Failed to create template.", err);
    return err;
  }
};
```

```
}  
};
```

Um das Beispiel auszuführen, geben Sie in der Befehlszeile Folgendes ein. Die Vorlage wird zu Amazon SES hinzugefügt.

```
node ses_createtemplate.js
```

Dieser Beispielcode finden Sie [hier auf GitHub](#).

Aktualisieren einer E-Mail-Vorlage

Verwenden Sie in diesem Beispiel ein Modul Node.js, um eine E-Mail-Vorlage für Amazon SES zu erstellen.

Erstellen Sie ein `libs` Verzeichnis und ein Modul Node.js mit dem Dateinamen `sesClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SES SES-Client-Objekt erstellt wird. **REGION** Ersetzen Sie es durch Ihre AWS Region.

```
import { SESClient } from "@aws-sdk/client-ses";  
// Set the AWS Region.  
const REGION = "us-east-1";  
// Create SES service object.  
const sesClient = new SESClient({ region: REGION });  
export { sesClient };
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ses_updatetemplate.js`. Konfigurieren Sie das SDK wie zuvor gezeigt, einschließlich der Installation der erforderlichen Clients und Pakete.

Erstellen Sie ein Objekt, um die Template-Parameterwerte, die Sie in der Vorlage aktualisieren möchten, mit dem erforderlichen `TemplateName`-Parameter an die `updateTemplateCommand`-Methode der SES-Client-Klasse zu übergeben. Um die `updateTemplateCommand` Methode aufzurufen, rufen Sie ein Amazon SES SES-Serviceobjekt auf und übergeben die Parameter.

Note

In diesem Beispiel werden die erforderlichen AWS Service V3-Paketclients und V3-Befehle importiert und verwendet und die `send` Methode in einem Async/Await-Muster verwendet. Sie

können dieses Beispiel stattdessen mit V2-Befehlen erstellen, indem Sie einige geringfügige Änderungen vornehmen. Details hierzu finden Sie unter [Verwenden von v3-Befehlen](#).

Note

TEMPLATE_NAME Ersetzen Sie es durch den Namen der Vorlage und durch den **HTML_PART** mit HTML markierten Inhalt der E-Mail.

```
import { UpdateTemplateCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "../libs/sesClient.js";

const TEMPLATE_NAME = getUniqueName("TemplateName");
const HTML_PART = "<h1>Hello, World!</h1>";

const createUpdateTemplateCommand = () => {
  return new UpdateTemplateCommand({
    Template: {
      TemplateName: TEMPLATE_NAME,
      HtmlPart: HTML_PART,
      SubjectPart: "Example",
      TextPart: "Updated template text.",
    },
  });
};

const run = async () => {
  const updateTemplateCommand = createUpdateTemplateCommand();

  try {
    return await sesClient.send(updateTemplateCommand);
  } catch (err) {
    console.log("Failed to update template.", err);
    return err;
  }
};
```

Um das Beispiel auszuführen, geben Sie in der Befehlszeile Folgendes ein. Amazon SES gibt die Vorlagendetails zurück.

```
node ses_updatetemplate.js
```

Dieser Beispielcode finden Sie [hier auf GitHub](#).

Löschen einer E-Mail-Vorlage

Verwenden Sie in diesem Beispiel ein Modul Node.js, um eine E-Mail-Vorlage für Amazon SES zu erstellen.

Erstellen Sie ein `libs` Verzeichnis und ein Modul Node.js mit dem Dateinamen `sesClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SES SES-Client-Objekt erstellt wird. **REGION** Ersetzen Sie es durch Ihre AWS Region.

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ses_deletetemplate.js`. Konfigurieren Sie das SDK wie zuvor gezeigt, einschließlich der Installation der erforderlichen Clients und Pakete.

Erstellen Sie ein Objekt, um den erforderlichen `TemplateName`-Parameter an die `DeleteTemplateCommand`-Methode der SES-Client-Klasse zu übergeben. Um die `DeleteTemplateCommand` Methode aufzurufen, rufen Sie ein Amazon SES SES-Serviceobjekt auf und übergeben die Parameter.

Note

In diesem Beispiel werden die erforderlichen AWS Service V3-Paketclients und V3-Befehle importiert und verwendet und die `send` Methode in einem Async/Await-Muster verwendet. Sie können dieses Beispiel stattdessen mit V2-Befehlen erstellen, indem Sie einige geringfügige Änderungen vornehmen. Details hierzu finden Sie unter [Verwenden von v3-Befehlen](#).

Note

TEMPLATE_NAME Ersetzen Sie es durch den Namen der Vorlage, die gelöscht werden soll.

```
import { DeleteTemplateCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "../libs/sesClient.js";

const TEMPLATE_NAME = getUniqueName("TemplateName");

const createDeleteTemplateCommand = (templateName) =>
  new DeleteTemplateCommand({ TemplateName: templateName });

const run = async () => {
  const deleteTemplateCommand = createDeleteTemplateCommand(TEMPLATE_NAME);

  try {
    return await sesClient.send(deleteTemplateCommand);
  } catch (err) {
    console.log("Failed to delete template.", err);
    return err;
  }
};
```

Um das Beispiel auszuführen, geben Sie in der Befehlszeile Folgendes ein. Amazon SES gibt die Vorlagendetails zurück.

```
node ses_deletetemplate.js
```

Dieser Beispielcode finden Sie [hier auf GitHub](#).

Senden von E-Mails mit Amazon SES



Dieses Node.js-Codebeispiel zeigt:

- das Senden einer Text- oder HTML-E-Mail.
- das Senden von E-Mails, die auf einer E-Mail-Vorlage basieren.
- das Senden von Massen-E-Mails, die auf einer E-Mail-Vorlage basieren.

Die Amazon SES SES-API bietet Ihnen zwei verschiedene Möglichkeiten, eine E-Mail zu senden, je nachdem, wie viel Kontrolle Sie über die Zusammensetzung der E-Mail-Nachricht haben möchten: formatiert und roh. Weitere Informationen finden Sie unter [Senden formatierter E-Mails mit der Amazon SES SES-API](#) und [Senden von Roh-E-Mails mit der Amazon SES SES-API](#).

Das Szenario

In diesem Beispiel verwenden Sie mehrere Node.js-Module, um E-Mails auf verschiedene Weisen zu senden. Die Module von Node.js verwenden das SDK JavaScript, um E-Mail-Vorlagen mithilfe der folgenden Methoden der SES Client-Klasse zu erstellen und zu verwenden:

- [SendEmailCommand](#)
- [SendTemplatedEmailCommand](#)
- [SendBulkTemplatedEmailCommand](#)

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Richten Sie die Projektumgebung ein, um diese TypeScript Node-Beispiele auszuführen, und installieren Sie die erforderlichen Module AWS SDK für JavaScript und Module von Drittanbietern. Folgen Sie den Anweisungen auf [GitHub](#).
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zur Bereitstellung einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter [Dateien mit gemeinsam genutzten Konfigurationen und Anmeldeinformationen](#) im AWS SDKs Referenzhandbuch zu Tools.

Important

Diese Beispiele zeigen, wie Client-Dienstobjekte und Befehle mithilfe von ECMAScript6 (ES6) importiert/exportiert werden.

- Dazu ist die Version 13.x von Node.js oder höher erforderlich. Informationen zum Herunterladen und Installieren der neuesten Version von Node.js finden Sie unter [Node.js downloads](#).
- Wenn Sie die CommonJS-Syntax bevorzugen, finden Sie weitere Informationen unter [JavaScript ES6/CommonJs-Syntax](#).

Anforderungen für den Versand von E-Mail-Nachrichten

Amazon SES verfasst eine E-Mail-Nachricht und stellt sie sofort in die Warteschlange für den Versand. Damit Sie mithilfe der `SendEmailCommand`-Methode E-Mails senden können, muss Ihre Nachricht die folgenden Anforderungen erfüllen:

- Sie müssen die Nachricht von einer verifizierten E-Mail-Adresse oder Domäne senden. Wenn Sie versuchen, E-Mails über eine nicht verifizierte Adresse oder Domäne zu senden, führt die Operation zu einem "Email address not verified"-Fehler.
- Wenn sich Ihr Konto noch in der Amazon SES Sandbox befindet, können Sie die E-Mails nur an verifizierte Adressen oder Domänen oder E-Mail-Adressen des Amazon SES-Postfachsimulators senden. Weitere Informationen finden Sie unter [Verifizieren von E-Mail-Adressen und Domains](#) im Amazon Simple Email Service Developer Guide.
- Die Gesamtgröße der Nachricht, einschließlich Anlagen, muss kleiner als 10 MB sein.
- Die Nachricht muss mindestens eine E-Mail-Adresse für einen Empfänger enthalten. Bei der Empfänger-Adresse kann es sich um eine Empfängeradresse, eine CC: Adresse oder BCC: Adresse handeln. Wenn die E-Mail-Adresse eines Empfängers nicht gültig ist (d. h., sie hat nicht das Format `UserName@[SubDomain.]Domain.TopLevelDomain`), wird die gesamte Nachricht zurückgewiesen, auch wenn die Nachricht andere gültige Empfänger enthält.
- Die Nachricht darf in den Feldern An:, CC: und BCC: nicht mehr als 50 Empfänger enthalten. Wenn Sie eine E-Mail an eine größere Zielgruppe senden möchten, können Sie Ihre Empfängerliste in Gruppen von höchstens 50 unterteilen und dann die `sendEmail`-Methode aufrufen, um die Nachricht mehrmals an die einzelnen Gruppen zu senden.

Eine E-Mail senden

In diesem Beispiel verwenden Sie ein Node.js-Modul zum Senden von E-Mail mit Amazon SES.

Erstellen Sie ein `libs` Verzeichnis und ein Modul `Node.js` mit dem Dateinamen `sesClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SES SES-Client-Objekt erstellt wird. **REGION** Ersetzen Sie es durch Ihre AWS Region.

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
export { sesClient };
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein `Node.js`-Modul mit dem Dateinamen `ses_sendemail.js`. Konfigurieren Sie das SDK wie zuvor gezeigt, einschließlich der Installation der erforderlichen Clients und Pakete.

Erstellen Sie ein Objekt, um die Parameterwerte, die die zu sendende E-Mail definieren, einschließlich Absender- und Empfängeradressen, Betreff und E-Mail-Text in Klartext- und HTML-Formaten, an die `SendEmailCommand` Methode der SES Client-Klasse zu übergeben. Um die `SendEmailCommand` Methode aufzurufen, rufen Sie ein Amazon SES SES-Serviceobjekt auf und übergeben die Parameter.

Note

In diesem Beispiel werden die erforderlichen AWS Service V3-Paketclients und V3-Befehle importiert und verwendet und die `send` Methode in einem `Async/Await`-Muster verwendet. Sie können dieses Beispiel stattdessen mit V2-Befehlen erstellen, indem Sie einige geringfügige Änderungen vornehmen. Details hierzu finden Sie unter [Verwenden von v3-Befehlen](#).

Note

toAddress Ersetzen Sie es durch die Adresse, an die die E-Mail gesendet werden soll, und **fromAddress** durch die E-Mail-Adresse, von der die E-Mail gesendet werden soll.

```
import { SendEmailCommand } from "@aws-sdk/client-ses";
import { sesClient } from "../libs/sesClient.js";
```

```
const createSendEmailCommand = (toAddress, fromAddress) => {
  return new SendEmailCommand({
    Destination: {
      /* required */
      CcAddresses: [
        /* more items */
      ],
      ToAddresses: [
        toAddress,
        /* more To-email addresses */
      ],
    },
    Message: {
      /* required */
      Body: {
        /* required */
        Html: {
          Charset: "UTF-8",
          Data: "HTML_FORMAT_BODY",
        },
        Text: {
          Charset: "UTF-8",
          Data: "TEXT_FORMAT_BODY",
        },
      },
      Subject: {
        Charset: "UTF-8",
        Data: "EMAIL_SUBJECT",
      },
    },
    Source: fromAddress,
    ReplyToAddresses: [
      /* more items */
    ],
  });
};

const run = async () => {
  const sendEmailCommand = createSendEmailCommand(
    "recipient@example.com",
    "sender@example.com",
  );

  try {
```

```
    return await sesClient.send(sendEmailCommand);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected } */
      const messageRejectedError = caught;
      return messageRejectedError;
    }
    throw caught;
  }
};
```

Um das Beispiel auszuführen, geben Sie in der Befehlszeile Folgendes ein. Die E-Mail befindet sich in der Warteschlange für den Versand durch Amazon SES.

```
node ses_sendemail.js
```

Diesen Beispielcode [finden Sie hier auf GitHub](#).

Senden einer E-Mail mithilfe einer Vorlage

In diesem Beispiel verwenden Sie ein Node.js-Modul zum Senden von E-Mail mit Amazon SES. Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ses_sendtemplatedemail.js`. Konfigurieren Sie das SDK wie zuvor gezeigt, einschließlich der Installation der erforderlichen Clients und Pakete.

Erstellen Sie ein Objekt, um die Parameterwerte, die die zu sendende E-Mail definieren, z. B. die Sender- und Empfängeradressen, Betreff, E-Mail-Text im Klartext- und HTML-Format, an die `SendTemplatedEmailCommand`-Methode der SES-Client-Klasse zu übergeben. Um die `SendTemplatedEmailCommand` Methode aufzurufen, rufen Sie ein Amazon SES SES-Client-Serviceobjekt auf und übergeben die Parameter.

Note

In diesem Beispiel werden die erforderlichen AWS Service V3-Paketclients und V3-Befehle importiert und verwendet und die `send` Methode in einem Async/Await-Muster verwendet. Sie können dieses Beispiel stattdessen mit V2-Befehlen erstellen, indem Sie einige geringfügige Änderungen vornehmen. Details hierzu finden Sie unter [Verwenden von v3-Befehlen](#).

 Note

REGION Ersetzen Sie es durch Ihre AWS Region, **USER** durch den Namen und die E-Mail-Adresse, an die die E-Mail gesendet werden soll, **VERIFIED_EMAIL** durch die E-Mail-Adresse, von der aus die E-Mail gesendet werden soll, und **TEMPLATE_NAME** durch den Namen der Vorlage.

```
import { SendTemplatedEmailCommand } from "@aws-sdk/client-ses";
import {
  getUniqueName,
  postfix,
} from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "../libs/sesClient.js";

/**
 * Replace this with the name of an existing template.
 */
const TEMPLATE_NAME = getUniqueName("ReminderTemplate");

/**
 * Replace these with existing verified emails.
 */
const VERIFIED_EMAIL = postfix(getUniqueName("Bilbo"), "@example.com");

const USER = { firstName: "Bilbo", emailAddress: VERIFIED_EMAIL };

/**
 *
 * @param { { emailAddress: string, firstName: string } } user
 * @param { string } templateName - The name of an existing template in Amazon SES.
 * @returns { SendTemplatedEmailCommand }
 */
const createReminderEmailCommand = (user, templateName) => {
  return new SendTemplatedEmailCommand({
    /**
     * Here's an example of how a template would be replaced with user data:
     * Template: <h1>Hello {{contact.firstName}},</h1><p>Don't forget about the party
    gifts!</p>
     * Destination: <h1>Hello Bilbo,</h1><p>Don't forget about the party gifts!</p>
     */
    Destination: { ToAddresses: [user.emailAddress] },
```

```
    TemplateData: JSON.stringify({ contact: { firstName: user.firstName } }),
    Source: VERIFIED_EMAIL,
    Template: templateName,
  });
};

const run = async () => {
  const sendReminderEmailCommand = createReminderEmailCommand(
    USER,
    TEMPLATE_NAME,
  );
  try {
    return await sesClient.send(sendReminderEmailCommand);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected } */
      const messageRejectedError = caught;
      return messageRejectedError;
    }
    throw caught;
  }
};
```

Um das Beispiel auszuführen, geben Sie in der Befehlszeile Folgendes ein. Die E-Mail befindet sich in der Warteschlange für den Versand durch Amazon SES.

```
node ses_sendtemplatedemail.js
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Senden von Massen-E-Mails mithilfe einer Vorlage

In diesem Beispiel verwenden Sie ein Node.js-Modul zum Senden von E-Mail mit Amazon SES.

Erstellen Sie ein `libs` Verzeichnis und ein Modul `Node.js` mit dem Dateinamen `sesClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SES SES-Client-Objekt erstellt wird. **REGION** Ersetzen Sie es durch Ihre AWS Region.

```
import { SESClient } from "@aws-sdk/client-ses";
// Set the AWS Region.
const REGION = "us-east-1";
// Create SES service object.
const sesClient = new SESClient({ region: REGION });
```

```
export { sesClient };
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `ses_sendbulktemplatedemail.js`. Konfigurieren Sie das SDK wie zuvor gezeigt, einschließlich der Installation der erforderlichen Clients und Pakete.

Erstellen Sie ein Objekt, um die Parameterwerte, die die zu sendende E-Mail definieren, einschließlich Absender- und Empfängeradressen, Betreff und E-Mail-Text in Klartext- und HTML-Formaten, an die `SendBulkTemplatedEmailCommand` Methode der SES Client-Klasse zu übergeben. Um die `SendBulkTemplatedEmailCommand` Methode aufzurufen, rufen Sie ein Amazon SES SES-Serviceobjekt auf und übergeben die Parameter.

Note

In diesem Beispiel werden die erforderlichen AWS Service V3-Paketclients und V3-Befehle importiert und verwendet und die `send` Methode in einem `Async/Await`-Muster verwendet. Sie können dieses Beispiel stattdessen mit V2-Befehlen erstellen, indem Sie einige geringfügige Änderungen vornehmen. Details hierzu finden Sie unter [Verwenden von v3-Befehlen](#).

Note

USERS Ersetzen Sie es durch die Namen und E-Mail-Adressen, an die die E-Mail gesendet werden soll, ***VERIFIED_EMAIL_1*** durch die E-Mail-Adresse, von der die E-Mail gesendet werden soll, und ***TEMPLATE_NAME*** durch den Namen der Vorlage.

```
import { SendBulkTemplatedEmailCommand } from "@aws-sdk/client-ses";
import {
  getUniqueName,
  postfix,
} from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "../libs/sesClient.js";

/**
 * Replace this with the name of an existing template.
 */
const TEMPLATE_NAME = getUniqueName("ReminderTemplate");
```

```

/**
 * Replace these with existing verified emails.
 */
const VERIFIED_EMAIL_1 = postfix(getUniqueName("Bilbo"), "@example.com");
const VERIFIED_EMAIL_2 = postfix(getUniqueName("Frodo"), "@example.com");

const USERS = [
  { firstName: "Bilbo", emailAddress: VERIFIED_EMAIL_1 },
  { firstName: "Frodo", emailAddress: VERIFIED_EMAIL_2 },
];

/**
 *
 * @param { { emailAddress: string, firstName: string }[] } users
 * @param { string } templateName the name of an existing template in SES
 * @returns { SendBulkTemplatedEmailCommand }
 */
const createBulkReminderEmailCommand = (users, templateName) => {
  return new SendBulkTemplatedEmailCommand({
    /**
     * Each 'Destination' uses a corresponding set of replacement data. We can map each
     user
     * to a 'Destination' and provide user specific replacement data to create
     personalized emails.
     *
     * Here's an example of how a template would be replaced with user data:
     * Template: <h1>Hello {{name}},</h1><p>Don't forget about the party gifts!</p>
     * Destination 1: <h1>Hello Bilbo,</h1><p>Don't forget about the party gifts!</p>
     * Destination 2: <h1>Hello Frodo,</h1><p>Don't forget about the party gifts!</p>
     */
    Destinations: users.map((user) => ({
      Destination: { ToAddresses: [user.emailAddress] },
      ReplacementTemplateData: JSON.stringify({ name: user.firstName }),
    })),
    DefaultTemplateData: JSON.stringify({ name: "Shireling" }),
    Source: VERIFIED_EMAIL_1,
    Template: templateName,
  });
};

const run = async () => {
  const sendBulkTemplateEmailCommand = createBulkReminderEmailCommand(
    USERS,

```



```
    TEMPLATE_NAME,  
  );  
  try {  
    return await sesClient.send(sendBulkTemplateEmailCommand);  
  } catch (caught) {  
    if (caught instanceof Error && caught.name === "MessageRejected") {  
      /** @type { import('@aws-sdk/client-ses').MessageRejected } */  
      const messageRejectedError = caught;  
      return messageRejectedError;  
    }  
    throw caught;  
  }  
};
```

Um das Beispiel auszuführen, geben Sie in der Befehlszeile Folgendes ein. Die E-Mail befindet sich in der Warteschlange für den Versand durch Amazon SES.

```
node ses_sendbulktemplatedemail.js
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Beispiele für Amazon Simple Notification Service

Amazon Simple Notification Service (Amazon SNS) ist ein Webservice, der die Zustellung oder den Versand von Nachrichten an abonnierte Endpunkte oder Kunden koordiniert und verwaltet.

In Amazon SNS gibt es zwei Arten von Kunden — Herausgeber und Abonnenten —, die auch als Produzenten und Verbraucher bezeichnet werden.



Herausgeber kommunizieren asynchron mit Abonnenten, indem sie eine Nachricht erstellen und an ein Thema senden, bei dem es sich wirklich um einen logischen Zugriffspunkt und

Kommunikationskanal handelt. Abonnenten (Webserver, E-Mail-Adressen, Amazon SQS-Warteschlangen, AWS Lambda Funktionen) konsumieren oder empfangen die Nachricht oder Benachrichtigung über eines der unterstützten Protokolle (Amazon SQS, HTTP/S, E-Mail, SMS AWS Lambda), wenn sie das Thema abonniert haben.

Die JavaScript API für Amazon SNS wird über die [Klasse: SNS](#) verfügbar gemacht.

Themen

- [Themen in Amazon SNS verwalten](#)
- [Nachrichten in Amazon SNS veröffentlichen](#)
- [Verwaltung von Abonnements in Amazon SNS](#)
- [Senden von SMS-Nachrichten mit Amazon SNS](#)

Themen in Amazon SNS verwalten



Dieses Node.js-Codebeispiel zeigt:

- So erstellen Sie Themen in Amazon SNS, zu denen Sie Benachrichtigungen veröffentlichen können.
- So löschen Sie in Amazon SNS erstellte Themen.
- Abrufen einer Liste von verfügbaren Themen
- Abrufen und Festlegen von Themenattributen

Das Szenario

In diesem Beispiel verwenden Sie eine Reihe von Node.js -Modulen, um Amazon SNS SNS-Themen zu erstellen, aufzulisten und zu löschen und Themenattribute zu verarbeiten. Die Module Node.js verwenden das SDK JavaScript zur Verwaltung von Themen mithilfe der folgenden Methoden der SNS Client-Klasse:

- [CreateTopicCommand](#)
- [ListTopicsCommand](#)

- [DeleteTopicCommand](#)
- [GetTopicAttributesCommand](#)
- [SetTopicAttributesCommand](#)

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Richten Sie die Projektumgebung ein, um diese TypeScript Node-Beispiele auszuführen, und installieren Sie die erforderlichen Module AWS SDK für JavaScript und Module von Drittanbietern. Folgen Sie den Anweisungen auf [GitHub](#).
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zur Bereitstellung einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter [Dateien mit gemeinsam genutzten Konfigurationen und Anmeldeinformationen](#) im AWS SDKs Referenzhandbuch zu Tools.

Important

Diese Beispiele veranschaulichen, wie Sie Objekte und Befehle mithilfe von ECMAScript6 (ES6) als import/export Client verwalten können.

- Dies erfordert Node.js Version 13.x oder höher. Informationen zum Herunterladen und Installieren der neuesten Version von Node.js finden Sie unter [Node.js downloads](#).
- Wenn Sie die CommonJS-Syntax bevorzugen, finden Sie weitere Informationen unter [JavaScript ES6/CommonJs-Syntax](#).

Erstellen eines Themas

Verwenden Sie in diesem Beispiel ein Modul Node.js, um ein Amazon SNS SNS-Thema zu erstellen.

Erstellen Sie ein `libs` Verzeichnis und ein Modul Node.js mit dem Dateinamen `snsClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SNS SNS-Client-Objekt erstellt wird. Ersetzen Sie es **REGION** durch Ihre AWS Region.

```
import { SNSClient } from "@aws-sdk/client-sns";
```

```
// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `create-topic.js`. Konfigurieren Sie das SDK wie zuvor gezeigt, einschließlich der Installation der erforderlichen Clients und Pakete.

Erstellen Sie ein Objekt, um den Namen (Name) für das neue Thema an die `CreateTopicCommand`-Methode der SNS-Client-Klasse zu übergeben. Um die `CreateTopicCommand` Methode aufzurufen, erstellen Sie eine asynchrone Funktion, die ein Amazon SNS-Serviceobjekt aufruft und das Parameterobjekt übergibt. Die data zurückgegebene enthält den ARN des Themas.

Note

Ersetze es ***TOPIC_NAME*** durch den Namen des Themas.

```
import { CreateTopicCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicName - The name of the topic to create.
 */
export const createTopic = async (topicName = "TOPIC_NAME") => {
  const response = await snsClient.send(
    new CreateTopicCommand({ Name: topicName }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '087b8ad2-4593-50c4-a496-d7e90b82cf3e',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:TOPIC_NAME'
  // }
```

```
// }  
return response;  
};
```

Um das Beispiel auszuführen, geben Sie in der Befehlszeile Folgendes ein.

```
node create-topic.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Auflisten Ihrer -Themen

Verwenden Sie in diesem Beispiel ein Modul Node.js, um alle Amazon SNS SNS-Themen aufzulisten.

Erstellen Sie ein `libs` Verzeichnis und ein Modul Node.js mit dem Dateinamen `snsClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SNS SNS-Client-Objekt erstellt wird. Ersetzen Sie es **REGION** durch Ihre AWS Region.

```
import { SNSClient } from "@aws-sdk/client-sns";  
  
// The AWS Region can be provided here using the `region` property. If you leave it  
// blank  
// the SDK will default to the region set in your AWS config.  
export const snsClient = new SNSClient({});
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `list-topics.js`. Konfigurieren Sie das SDK wie zuvor gezeigt, einschließlich der Installation der erforderlichen Clients und Pakete.

Erstellen Sie ein leeres Objekt, das an die `ListTopicsCommand`-Methode der SNS-Client-Klasse übergeben wird. Um die `ListTopicsCommand` Methode aufzurufen, erstellen Sie eine asynchrone Funktion, die ein Amazon SNS-Serviceobjekt aufruft und das Parameterobjekt übergibt. Die data zurückgegebene Datei enthält ein Array Ihres Themas Amazon Resource Names (ARNs).

```
import { ListTopicsCommand } from "@aws-sdk/client-sns";  
import { snsClient } from "../libs/snsClient.js";  
  
export const listTopics = async () => {
```

```
const response = await snsClient.send(new ListTopicsCommand({}));
console.log(response);
// {
//   '$metadata': {
//     httpStatusCode: 200,
//     requestId: '936bc5ad-83ca-53c2-b0b7-9891167b909e',
//     extendedRequestId: undefined,
//     cfId: undefined,
//     attempts: 1,
//     totalRetryDelay: 0
//   },
//   Topics: [ { TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:mytopic' } ]
// }
return response;
};
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node list-topics.js
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Löschen eines Themas

Verwenden Sie in diesem Beispiel ein Modul Node.js, um ein Amazon SNS SNS-Thema zu löschen.

Erstellen Sie ein `libs` Verzeichnis und ein Modul Node.js mit dem Dateinamen `snsClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SNS SNS-Client-Objekt erstellt wird. Ersetzen Sie es **REGION** durch Ihre AWS Region.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `delete-topic.js`. Konfigurieren Sie das SDK wie zuvor gezeigt, einschließlich der Installation der erforderlichen Clients und Pakete.

Erstellen Sie ein Objekt, das den TopicArn des zu löschenden Themas enthält, um es an die DeleteTopicCommand-Methode der SNS-Client-Klasse zu übergeben. Um die DeleteTopicCommand Methode aufzurufen, erstellen Sie eine asynchrone Funktion, die ein Amazon SNS SNS-Client-Serviceobjekt aufruft und das Parameter-Objekt übergibt.

 Note

TOPIC_ARN Ersetzen Sie es durch den Amazon-Ressourcennamen (ARN) des Themas, das Sie löschen möchten.

```
import { DeleteTopicCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic to delete.
 */
export const deleteTopic = async (topicArn = "TOPIC_ARN") => {
  const response = await snsClient.send(
    new DeleteTopicCommand({ TopicArn: topicArn }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'a10e2886-5a8f-5114-af36-75bd39498332',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
};
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node delete-topic.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Abrufen von Themenattributen

Verwenden Sie in diesem Beispiel ein Modul Node.js, um Attribute eines Amazon SNS SNS-Themas abzurufen.

Erstellen Sie ein `libs` Verzeichnis und ein Modul Node.js mit dem Dateinamen `snsClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SNS SNS-Client-Objekt erstellt wird. Ersetzen Sie es **REGION** durch Ihre AWS Region.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `get-topic-attributes.js`. Konfigurieren Sie das SDK wie zuvor dargestellt.

Erstellen Sie ein Objekt, das den TopicArn eines zu löschenden Themas enthält, um es an die `GetTopicAttributesCommand`-Methode der SNS-Client-Klasse zu übergeben. Um die `GetTopicAttributesCommand` Methode aufzurufen, wird ein Amazon SNS SNS-Client-Serviceobjekt aufgerufen und das Parameter-Objekt übergeben.

Note

Ersetze es **TOPIC_ARN** durch den ARN des Themas.

```
import { GetTopicAttributesCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic to retrieve attributes for.
 */
export const getTopicAttributes = async (topicArn = "TOPIC_ARN") => {
  const response = await snsClient.send(
    new GetTopicAttributesCommand({
      TopicArn: topicArn,
```



```

    }},
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '36b6a24e-5473-5d4e-ac32-ff72d9a73d94',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Attributes: {
  //     Policy: '{...}',
  //     Owner: 'xxxxxxxxxxxxx',
  //     SubscriptionsPending: '1',
  //     TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxxx:mytopic',
  //     TracingConfig: 'PassThrough',
  //     EffectiveDeliveryPolicy: '{"http":{"defaultHealthyRetryPolicy":
{"minDelayTarget":20,"maxDelayTarget":20,"numRetries":3,"numMaxDelayRetries":0,"numNoDelayRetri
{"headerContentType":"text/plain; charset=UTF-8"}}}',
  //     SubscriptionsConfirmed: '0',
  //     DisplayName: '',
  //     SubscriptionsDeleted: '1'
  //   }
  // }
  return response;
};

```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node get-topic-attributes.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Festlegen von Themenattributen

Verwenden Sie in diesem Beispiel ein Modul Node.js, um die veränderbaren Attribute eines Amazon SNS SNS-Themas festzulegen.

Erstellen Sie ein `libs` Verzeichnis und ein Modul Node.js mit dem Dateinamen `snsClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SNS SNS-Client-Objekt erstellt wird. Ersetzen Sie es **REGION** durch Ihre AWS Region.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `set-topic-attributes.js`. Konfigurieren Sie das SDK wie zuvor dargestellt.

Erstellen Sie ein Objekt, das die Parameter für eine Aktualisierung des Attributs enthält. Dazu gehören der TopicArn des Themas, dessen Attribute Sie festlegen möchten, der Name des festzulegenden Attributs und der neue Wert für dieses Attribut. Sie können nur die Attribute Policy, DisplayName und DeliveryPolicy festlegen. Übergeben Sie die Parameter an die `SetTopicAttributesCommand`-Methode der Client-Klasse SNS. Um die `SetTopicAttributesCommand` Methode aufzurufen, erstellen Sie eine asynchrone Funktion, die ein Amazon SNS SNS-Client-Serviceobjekt aufruft und das Parameter-Objekt übergibt.

Note

ATTRIBUTE_NAME Ersetzen Sie durch den Namen des Attributs, das Sie festlegen, **TOPIC_ARN** durch den Amazon-Ressourcennamen (ARN) des Themas, dessen Attribute Sie festlegen möchten, und **NEW_ATTRIBUTE_VALUE** durch den neuen Wert für dieses Attribut.

```
import { SetTopicAttributesCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

export const setTopicAttributes = async (
  topicArn = "TOPIC_ARN",
  attributeName = "DisplayName",
  attributeValue = "Test Topic",
) => {
  const response = await snsClient.send(
    new SetTopicAttributesCommand({
      AttributeName: attributeName,
      AttributeValue: attributeValue,
      TopicArn: topicArn,
```

```
    }},  
  );  
  console.log(response);  
  // {  
  //   '$metadata': {  
  //     httpStatusCode: 200,  
  //     requestId: 'd1b08d0e-e9a4-54c3-b8b1-d03238d2b935',  
  //     extendedRequestId: undefined,  
  //     cfId: undefined,  
  //     attempts: 1,  
  //     totalRetryDelay: 0  
  //   }  
  // }  
  return response;  
};
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node set-topic-attributes.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Nachrichten in Amazon SNS veröffentlichen



Dieses Node.js-Codebeispiel zeigt:

- So veröffentlichen Sie Nachrichten zu einem Amazon SNS SNS-Thema.

Das Szenario

In diesem Beispiel verwenden Sie eine Reihe von Node.js -Modulen, um Nachrichten von Amazon SNS an Themenendpunkte, E-Mails oder Telefonnummern zu veröffentlichen. Die Module Node.js verwenden das SDK JavaScript , um Nachrichten mit dieser Methode der SNS Client-Klasse zu senden:

- [PublishCommand](#)

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Richten Sie die Projektumgebung ein, um diese TypeScript Node-Beispiele auszuführen, und installieren Sie die erforderlichen Module AWS SDK für JavaScript und Module von Drittanbietern. Folgen Sie den Anweisungen auf [GitHub](#).
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zur Bereitstellung einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter [Dateien mit gemeinsam genutzten Konfigurationen und Anmeldeinformationen](#) im AWS SDKs Referenzhandbuch zu Tools.

Important

Diese Beispiele veranschaulichen, wie Sie Objekte und Befehle mithilfe von ECMAScript6 (ES6) als import/export Client verwalten können.

- Dies erfordert Node.js Version 13.x oder höher. Informationen zum Herunterladen und Installieren der neuesten Version von Node.js finden Sie unter [Node.js downloads](#).
- Wenn Sie die CommonJS-Syntax bevorzugen, finden Sie weitere Informationen unter [JavaScript ES6/CommonJs-Syntax](#).

Veröffentlichen einer Nachricht in einem SNS-Thema

Verwenden Sie in diesem Beispiel ein Modul Node.js, um eine Nachricht zu einem Amazon SNS SNS-Thema zu veröffentlichen.

Erstellen Sie ein `libs` Verzeichnis und ein Modul Node.js mit dem Dateinamen `snsClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SNS SNS-Client-Objekt erstellt wird. Ersetzen Sie es **REGION** durch Ihre AWS Region.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `publish-topic.js`. Konfigurieren Sie das SDK wie zuvor dargestellt.

Erstellen Sie ein Objekt, das die Parameter für die Veröffentlichung einer Nachricht enthält, einschließlich des Nachrichtentexts und des Amazon-Ressourcennamens (ARN) des Amazon SNS-Topics. Einzelheiten zu den verfügbaren SMS-Attributen finden Sie unter [Set SMSAttributes](#).

Übergeben Sie die Parameter an die `PublishCommand` Methode der SNS Client-Klasse. Erstellen Sie eine asynchrone Funktion, die ein Amazon SNS SNS-Client-Serviceobjekt aufruft und das Parameterobjekt übergibt.

 Note

`MESSAGE_TEXT` Ersetzen Sie durch den Nachrichtentext und `TOPIC_ARN` durch den ARN des SNS-Themas.

```
import { PublishCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string | Record<string, any>} message - The message to send. Can be a plain
 * string or an object
 *
 *                                     if you are using the `json`
 * `MessageStructure`.
 * @param {string} topicArn - The ARN of the topic to which you would like to publish.
 */
export const publish = async (
  message = "Hello from SNS!",
  topicArn = "TOPIC_ARN",
) => {
  const response = await snsClient.send(
    new PublishCommand({
      Message: message,
      TopicArn: topicArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
```

```
//    httpStatusCode: 200,  
//    requestId: 'e7f77526-e295-5325-9ee4-281a43ad1f05',  
//    extendedRequestId: undefined,  
//    cfId: undefined,  
//    attempts: 1,  
//    totalRetryDelay: 0  
//  },  
//  MessageId: 'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxxx'  
// }  
return response;  
};
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node publish-topic.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Verwaltung von Abonnements in Amazon SNS



Dieses Node.js-Codebeispiel zeigt:

- So listen Sie alle Abonnements für ein Amazon SNS SNS-Thema auf.
- So abonnieren Sie eine E-Mail-Adresse, einen Anwendungsendpunkt oder eine AWS Lambda Funktion für ein Amazon SNS SNS-Thema.
- So melden Sie sich von Amazon SNS SNS-Themen ab.

Das Szenario

In diesem Beispiel verwenden Sie eine Reihe von Node.js -Modulen, um Benachrichtigungen zu Amazon SNS SNS-Themen zu veröffentlichen. Die Module Node.js verwenden das SDK JavaScript zur Verwaltung von Themen mithilfe der folgenden Methoden der SNS Client-Klasse:

- [ListSubscriptionsByTopicCommand](#)
- [SubscribeCommand](#)

- [ConfirmSubscriptionCommand](#)
- [UnsubscribeCommand](#)

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Richten Sie die Projektumgebung ein, um diese TypeScript Node-Beispiele auszuführen, und installieren Sie die erforderlichen Module AWS SDK für JavaScript und Module von Drittanbietern. Folgen Sie den Anweisungen auf [GitHub](#).
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zur Bereitstellung einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter [Dateien mit gemeinsam genutzten Konfigurationen und Anmeldeinformationen](#) im AWS SDKs Referenzhandbuch zu Tools.

Important

Diese Beispiele veranschaulichen, wie Sie Objekte und Befehle mithilfe von ECMAScript6 (ES6) als import/export Client verwalten können.

- Dies erfordert Node.js Version 13.x oder höher. Informationen zum Herunterladen und Installieren der neuesten Version von Node.js finden Sie unter [Node.js downloads](#).
- Wenn Sie die CommonJS-Syntax bevorzugen, finden Sie weitere Informationen unter [JavaScript ES6/CommonJs-Syntax](#).

Auflisten von Abonnements eines Themas

Verwenden Sie in diesem Beispiel ein Modul Node.js, um alle Abonnements für ein Amazon SNS SNS-Thema aufzulisten.

Erstellen Sie ein `libs` Verzeichnis und ein Modul Node.js mit dem Dateinamen `snsClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SNS SNS-Client-Objekt erstellt wird. Ersetzen Sie es **REGION** durch Ihre AWS Region.

```
import { SNSClient } from "@aws-sdk/client-sns";
```

```
// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `list-subscriptions-by-topic.js`. Konfigurieren Sie das SDK wie zuvor dargestellt.

Erstellen Sie ein Objekt, das den `TopicArn`-Parameter für das Thema enthält, dessen Abonnements Sie auflisten möchten. Übergeben Sie die Parameter an die `ListSubscriptionsByTopicCommand`-Methode der Client-Klasse `SNS`. Um die `ListSubscriptionsByTopicCommand` Methode aufzurufen, erstellen Sie eine asynchrone Funktion, die ein Amazon SNS `SNS-Client-Service`objekt aufruft und das Parameter-Objekt übergibt.

Note

`TOPIC_ARN` Ersetzen Sie es durch den Amazon-Ressourcennamen (ARN) für das Thema, dessen Abonnements Sie auflisten möchten.

```
import { ListSubscriptionsByTopicCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic for which you wish to list
 * subscriptions.
 */
export const listSubscriptionsByTopic = async (topicArn = "TOPIC_ARN") => {
  const response = await snsClient.send(
    new ListSubscriptionsByTopicCommand({ TopicArn: topicArn }),
  );

  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '0934fedf-0c4b-572e-9ed2-a3e38fadb0c8',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
```



```
//      attempts: 1,
//      totalRetryDelay: 0
//    },
//    Subscriptions: [
//      {
//        SubscriptionArn: 'PendingConfirmation',
//        Owner: '901487484989',
//        Protocol: 'email',
//        Endpoint: 'corepyle@amazon.com',
//        TopicArn: 'arn:aws:sns:us-east-1:901487484989:mytopic'
//      }
//    ]
//  }
return response;
};
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node list-subscriptions-by-topic.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Abonnieren eines Themas durch Hinterlegen einer E-Mail-Adresse

Verwenden Sie in diesem Beispiel ein Modul Node.js, um eine E-Mail-Adresse zu abonnieren, sodass sie SMTP-E-Mail-Nachrichten von einem Amazon SNS SNS-Thema empfängt.

Erstellen Sie ein `libs` Verzeichnis und ein Modul Node.js mit dem Dateinamen `snsClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SNS SNS-Client-Objekt erstellt wird. Ersetzen Sie es **REGION** durch Ihre AWS Region.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `subscribe-email.js`. Konfigurieren Sie das SDK wie zuvor dargestellt.

Erstellen Sie ein Objekt, das den Protocol-Parameter enthält, um das email-Protokoll, den TopicArn für das Thema, das abonniert werden soll, und eine E-Mail-Adresse als Endpoint der Nachricht anzugeben. Übergeben Sie die Parameter an die SubscribeCommand-Methode der Client-Klasse SNS. Sie können die subscribe Methode verwenden, um mehrere verschiedene Endpunkte für ein Amazon SNS SNS-Thema zu abonnieren, abhängig von den Werten, die für die übergebenen Parameter verwendet werden, wie andere Beispiele in diesem Thema zeigen werden.

Um die SubscribeCommand Methode aufzurufen, erstellen Sie eine asynchrone Funktion, die ein Amazon SNS SNS-Client-Serviceobjekt aufruft und das Parameter-Objekt übergibt.

Note

TOPIC_ARN Ersetzen Sie es durch den Amazon-Ressourcennamen (ARN) für das Thema und **EMAIL_ADDRESS** durch die E-Mail-Adresse, die Sie abonnieren möchten.

```
import { SubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic for which you wish to confirm a
 * subscription.
 * @param {string} emailAddress - The email address that is subscribed to the topic.
 */
export const subscribeEmail = async (
  topicArn = "TOPIC_ARN",
  emailAddress = "usern@me.com",
) => {
  const response = await snsClient.send(
    new SubscribeCommand({
      Protocol: "email",
      TopicArn: topicArn,
      Endpoint: emailAddress,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'c8e35bcd-b3c0-5940-9f66-06f6fcc108f0',
```

```
//    extendedRequestId: undefined,  
//    cfId: undefined,  
//    attempts: 1,  
//    totalRetryDelay: 0  
//  },  
//  SubscriptionArn: 'pending confirmation'  
// }  
};
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node subscribe-email.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Bestätigen von Abonnements

Verwenden Sie in diesem Beispiel ein Modul Node.js, um die Absicht eines Endpunktbesitzers zu überprüfen, E-Mails zu empfangen, indem Sie das Token validieren, das durch eine vorherige Abonnement-Aktion an den Endpunkt gesendet wurde.

Erstellen Sie ein `libs` Verzeichnis und ein Modul Node.js mit dem Dateinamen `snsClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SNS SNS-Client-Objekt erstellt wird. Ersetzen Sie es **REGION** durch Ihre AWS Region.

```
import { SNSClient } from "@aws-sdk/client-sns";  
  
// The AWS Region can be provided here using the `region` property. If you leave it  
// blank  
// the SDK will default to the region set in your AWS config.  
export const snsClient = new SNSClient({});
```

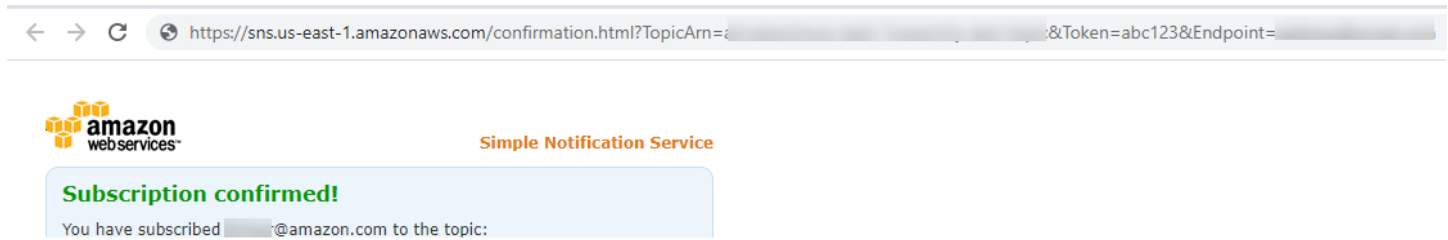
Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `confirm-subscription.js`. Konfigurieren Sie das SDK wie zuvor gezeigt, einschließlich der Installation der erforderlichen Clients und Pakete.

Definieren Sie die Parameter, einschließlich `TOPIC_ARN` und `TOKEN`, und definieren Sie einen Wert von `TRUE` oder `FALSE` für `AuthenticateOnUnsubscribe`.

Das Token ist ein kurzlebiges Token, das während einer vorherigen `SUBSCRIBE` Aktion an den Besitzer eines Endpunkts gesendet wurde. Bei einem E-Mail-Endpunkt `TOKEN` steht es

beispielsweise in der URL der E-Mail zur Bestätigung des Abonnements, die an den E-Mail-Besitzer gesendet wurde. Zum Beispiel abc123 ist das Token in der folgenden URL.



Um die `ConfirmSubscriptionCommand` Methode aufzurufen, erstellen Sie eine asynchrone Funktion, die ein Amazon SNS `SNS-Client-Serviceobjekt` aufruft und das Parameter-Objekt übergibt.

Note

TOPIC_ARN Ersetzen Sie es durch den Amazon-Ressourcennamen (ARN) für das Thema, **TOKEN** durch den Token-Wert aus der URL, die in einer früheren `Subscribe` Aktion an den Endpoint-Besitzer gesendet wurde, und definiere **AuthenticateOnUnsubscribe** mit dem Wert `TRUE` oder `FALSE`.

```
import { ConfirmSubscriptionCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} token - This token is sent the subscriber. Only subscribers
 *                        that are not AWS services (HTTP/S, email) need to be
 *                        confirmed.
 * @param {string} topicArn - The ARN of the topic for which you wish to confirm a
 *                            subscription.
 */
export const confirmSubscription = async (
  token = "TOKEN",
  topicArn = "TOPIC_ARN",
) => {
  const response = await snsClient.send(
    // A subscription only needs to be confirmed if the endpoint type is
    // HTTP/S, email, or in another AWS account.
    new ConfirmSubscriptionCommand({
      Token: token,
      TopicArn: topicArn,
      // If this is true, the subscriber cannot unsubscribe while unauthenticated.
    })
  );
}
```

```

        AuthenticateOnUnsubscribe: "false",
    }},
);
console.log(response);
// {
//   '$metadata': {
//     httpStatusCode: 200,
//     requestId: '4bb5bce9-805a-5517-8333-e1d2cface90b',
//     extendedRequestId: undefined,
//     cfId: undefined,
//     attempts: 1,
//     totalRetryDelay: 0
//   },
//   SubscriptionArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:TOPIC_NAME:xxxxxxxx-xxxx-
xxxx-xxxx-xxxxxxxxxxxxx'
// }
return response;
};

```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node confirm-subscription.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Abonnieren eines Themas mit einem Anwendungsendpunkt

Verwenden Sie in diesem Beispiel ein Modul Node.js, um einen mobilen Anwendungsendpunkt zu abonnieren, sodass dieser Benachrichtigungen von einem Amazon SNS SNS-Thema empfängt.

Erstellen Sie ein `libs` Verzeichnis und ein Modul Node.js mit dem Dateinamens `snsClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SNS SNS-Client-Objekt erstellt wird. Ersetzen Sie es **REGION** durch Ihre AWS Region.

```

import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});

```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `subscribe-app.js`. Konfigurieren Sie das SDK wie zuvor gezeigt, einschließlich der Installation der erforderlichen Module und Pakete.

Erstellen Sie ein Objekt, das den `Protocol` Parameter zur Angabe des application Protokolls, das `TopicArn` für das zu abonnierende Thema und den Amazon-Ressourcennamen (ARN) eines mobilen Anwendungsendpunkts für den `Endpoint` Parameter enthält. Übergeben Sie die Parameter an die `SubscribeCommand`-Methode der Client-Klasse `SNS`.

Um die `SubscribeCommand` Methode aufzurufen, erstellen Sie eine asynchrone Funktion, die ein Amazon SNS-Serviceobjekt aufruft und das Parameterobjekt übergibt.

Note

`TOPIC_ARN` Ersetzen Sie es durch den Amazon-Ressourcennamen (ARN) für das Thema und **`MOBILE_ENDPOINT_ARN`** durch den Endpunkt, für den Sie das Thema abonnieren.

```
import { SubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic the subscriber is subscribing to.
 * @param {string} endpoint - The Endpoint ARN of an application. This endpoint is
 *                           created
 *                           when an application registers for notifications.
 */
export const subscribeApp = async (
  topicArn = "TOPIC_ARN",
  endpoint = "ENDPOINT",
) => {
  const response = await snsClient.send(
    new SubscribeCommand({
      Protocol: "application",
      TopicArn: topicArn,
      Endpoint: endpoint,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
```

```
//    requestId: 'c8e35bcd-b3c0-5940-9f66-06f6fcc108f0',  
//    extendedRequestId: undefined,  
//    cfId: undefined,  
//    attempts: 1,  
//    totalRetryDelay: 0  
//  },  
//  SubscriptionArn: 'pending confirmation'  
// }  
return response;  
};
```

Um das Beispiel auszuführen, geben Sie in der Befehlszeile Folgendes ein.

```
node subscribe-app.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Abonnieren einer Lambda-Funktion für ein Thema

Verwenden Sie in diesem Beispiel ein Modul Node.js, um eine AWS Lambda Funktion zu abonnieren, sodass sie Benachrichtigungen von einem Amazon SNS SNS-Thema empfängt.

Erstellen Sie ein `libs` Verzeichnis und ein Modul Node.js mit dem Dateinamen `snsClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SNS SNS-Client-Objekt erstellt wird. Ersetzen Sie es **REGION** durch Ihre AWS Region.

```
import { SNSClient } from "@aws-sdk/client-sns";  
  
// The AWS Region can be provided here using the `region` property. If you leave it  
// blank  
// the SDK will default to the region set in your AWS config.  
export const snsClient = new SNSClient({});
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `subscribe-lambda.js`. Konfigurieren Sie das SDK wie zuvor dargestellt.

Erstellen Sie ein Objekt, das den `Protocol` Parameter enthält und `lambda` das Protokoll, das `TopicArn` für das zu abonnierende Thema und den Amazon-Ressourcennamen (ARN) einer

AWS Lambda Funktion als Endpoint Parameter angibt. Übergeben Sie die Parameter an die SubscribeCommand-Methode der Client-Klasse SNS.

Um die SubscribeCommand Methode aufzurufen, erstellen Sie eine asynchrone Funktion, die ein Amazon SNS SNS-Client-Serviceobjekt aufruft und das Parameter-Objekt übergibt.

 Note

TOPIC_ARN Ersetzen Sie durch den Amazon-Ressourcennamen (ARN) für das Thema und **LAMBDA_FUNCTION_ARN** durch den Amazon-Ressourcennamen (ARN) der Lambda-Funktion.

```
import { SubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic the subscriber is subscribing to.
 * @param {string} endpoint - The Endpoint ARN of and AWS Lambda function.
 */
export const subscribeLambda = async (
  topicArn = "TOPIC_ARN",
  endpoint = "ENDPOINT",
) => {
  const response = await snsClient.send(
    new SubscribeCommand({
      Protocol: "lambda",
      TopicArn: topicArn,
      Endpoint: endpoint,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'c8e35bcd-b3c0-5940-9f66-06f6fcc108f0',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'pending confirmation'
  // }
```



```
// }  
return response;  
};
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node subscribe-lambda.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Abmelden von einem Thema

Verwenden Sie in diesem Beispiel ein Modul Node.js, um ein Amazon SNS SNS-Themenabonnement zu kündigen.

Erstellen Sie ein `libs` Verzeichnis und ein Modul Node.js mit dem Dateinamen `snsClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SNS SNS-Client-Objekt erstellt wird. Ersetzen Sie es **REGION** durch Ihre AWS Region.

```
import { SNSClient } from "@aws-sdk/client-sns";  
  
// The AWS Region can be provided here using the `region` property. If you leave it  
// blank  
// the SDK will default to the region set in your AWS config.  
export const snsClient = new SNSClient({});
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `unsubscribe.js`. Konfigurieren Sie das SDK wie zuvor gezeigt, einschließlich der Installation der erforderlichen Clients und Pakete.

Erstellen Sie ein Objekt, das den `SubscriptionArn` Parameter enthält und den Amazon-Ressourcennamen (ARN) des Abonnements angibt, das gekündigt werden soll. Übergeben Sie die Parameter an die `UnsubscribeCommand`-Methode der Client-Klasse SNS.

Um die `UnsubscribeCommand` Methode aufzurufen, erstellen Sie eine asynchrone Funktion, die ein Amazon SNS SNS-Client-Serviceobjekt aufruft und das Parameter-Objekt übergibt.

 Note

TOPIC_SUBSCRIPTION_ARN Ersetzen Sie es durch den Amazon-Ressourcennamen (ARN) des Abonnements, um das Abonnement zu kündigen.

```
import { UnsubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} subscriptionArn - The ARN of the subscription to cancel.
 */
const unsubscribe = async (
  subscriptionArn = "arn:aws:sns:us-east-1:xxxxxxxxxxxx:mytopic:xxxxxxxx-xxxx-xxxx-
  xxxx-xxxxxxxxxxxx",
) => {
  const response = await snsClient.send(
    new UnsubscribeCommand({
      SubscriptionArn: subscriptionArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '0178259a-9204-507c-b620-78a7570a44c6',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
  return response;
};
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node unsubscribe.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Senden von SMS-Nachrichten mit Amazon SNS



Dieses Node.js-Codebeispiel zeigt:

- So rufen Sie SMS-Nachrichteneinstellungen für Amazon SNS ab und legen sie fest.
- Überprüfen, ob eine Telefonnummer vom Empfang von SMS-Nachrichten abgemeldet wurde
- Abrufen einer Liste der Telefonnummern, die vom Empfang von SMS-Nachrichten abgemeldet wurden
- Senden einer SMS-Nachricht

Das Szenario

Verwenden Sie Amazon SNS, um Textnachrichten oder SMS-Nachrichten an SMS-fähige Geräte zu senden. Sie können eine Nachricht direkt an eine Telefonnummer senden oder Sie können eine Nachricht an mehrere Telefonnummern gleichzeitig senden, indem Sie das Thema für diese Telefonnummern abonnieren und die Nachricht an das Thema senden.

In diesem Beispiel verwenden Sie eine Reihe von Node.js -Modulen, um SMS-Textnachrichten von Amazon SNS auf SMS-fähigen Geräten zu veröffentlichen. Die Module Node.js verwenden das SDK für JavaScript die Veröffentlichung von SMS-Nachrichten mit den folgenden Methoden der SNS Client-Klasse:

- [GetSMSAttributesCommand](#)
- [SetSMSAttributesCommand](#)
- [CheckIfPhoneNumberIsOptedOutCommand](#)
- [ListPhoneNumbersOptedOutCommand](#)
- [PublishCommand](#)

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Richten Sie die Projektumgebung ein, um diese TypeScript Node-Beispiele auszuführen, und installieren Sie die erforderlichen Module AWS SDK für JavaScript und Module von Drittanbietern. Folgen Sie den Anweisungen auf [GitHub](#).
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zur Bereitstellung einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter [Dateien mit gemeinsam genutzten Konfigurationen und Anmeldeinformationen](#) im AWS SDKs Referenzhandbuch zu Tools.

Important

Diese Beispiele veranschaulichen, wie Sie Objekte und Befehle mithilfe von ECMAScript6 (ES6) als import/export Client verwalten können.

- Dies erfordert Node.js Version 13.x oder höher. Informationen zum Herunterladen und Installieren der neuesten Version von Node.js finden Sie unter [Node.js downloads](#).
- Wenn Sie die CommonJS-Syntax bevorzugen, finden Sie weitere Informationen unter [JavaScript ES6/CommonJs-Syntax](#).

Abrufen von SMS-Attributen

Verwenden Sie Amazon SNS, um Einstellungen für SMS-Nachrichten festzulegen, z. B. wie Ihre Lieferungen optimiert werden (aus Kostengründen oder für eine zuverlässige Zustellung), Ihr monatliches Ausgabenlimit, wie Nachrichtenzustellungen protokolliert werden und ob Sie tägliche SMS-Nutzungsberichte abonnieren möchten. Diese Einstellungen werden abgerufen und als SMS-Attribute für Amazon SNS festgelegt.

Verwenden Sie in diesem Beispiel ein Modul Node.js, um die aktuellen SMS-Attribute in Amazon SNS abzurufen.

Erstellen Sie ein `libs` Verzeichnis und ein Modul Node.js mit dem Dateinamen `snsClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SNS SNS-Client-Objekt erstellt wird. Ersetzen Sie es **REGION** durch Ihre AWS Region.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
blank
```

```
// the SDK will default to the region set in your AWS config.  
export const snsClient = new SNSClient({});
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `get-sms-attributes.js`.

Konfigurieren Sie das SDK wie zuvor gezeigt, einschließlich des Herunterladens der erforderlichen Clients und Pakete. Erstellen Sie ein Objekt, das die Parameter zum Abrufen von SMS-Attributen enthält, einschließlich der Namen der einzelnen Attribute, die abgerufen werden. Einzelheiten zu verfügbaren SMS-Attributen finden Sie unter [Set SMSAttributes](#) in der Amazon Simple Notification Service API-Referenz.

In diesem Beispiel wird das `DefaultSMSType`-Attribut abgerufen. Dieses Attribut steuert, ob SMS-Nachrichten als `Promotional` oder als `Transactional` gesendet werden. Im ersten Fall wird die Nachrichtenzustellung im Hinblick auf die Kosten und im zweiten Fall im Hinblick auf höchste Zuverlässigkeit optimiert. Übergeben Sie die Parameter an die `SetTopicAttributesCommand`-Methode der Client-Klasse `SNS`. Um die `SetSMSAttributesCommand` Methode aufzurufen, erstellen Sie eine asynchrone Funktion, die ein Amazon SNS `SNS-Client-Serviceobjekt` aufruft und das Parameter-Objekt übergibt.

Note

Ersetzen Sie es `ATTRIBUTE_NAME` durch den Namen des Attributs.

```
import { GetSMSAttributesCommand } from "@aws-sdk/client-sns";  
import { snsClient } from "../libs/snsClient.js";  
  
export const getSmsAttributes = async () => {  
  const response = await snsClient.send(  
    // If you have not modified the account-level mobile settings of SNS,  
    // the DefaultSMSType is undefined. For this example, it was set to  
    // Transactional.  
    new GetSMSAttributesCommand({ attributes: ["DefaultSMSType"] }),  
  );  
  
  console.log(response);  
  // {  
  //   '$metadata': {  
  //     httpStatusCode: 200,
```

```
//    requestId: '67ad8386-4169-58f1-bdb9-debd281d48d5',
//    extendedRequestId: undefined,
//    cfId: undefined,
//    attempts: 1,
//    totalRetryDelay: 0
//  },
//  attributes: { DefaultSMSType: 'Transactional' }
// }
return response;
};
```

Um das Beispiel auszuführen, geben Sie in der Befehlszeile Folgendes ein.

```
node get-sms-attributes.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Festlegen von SMS-Attributen

Verwenden Sie in diesem Beispiel ein Modul Node.js, um die aktuellen SMS-Attribute in Amazon SNS abzurufen.

Erstellen Sie ein `libs` Verzeichnis und ein Modul Node.js mit dem Dateinamen `snsClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SNS SNS-Client-Objekt erstellt wird. Ersetzen Sie es **REGION** durch Ihre AWS Region.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `set-sms-attribute-type.js`. Konfigurieren Sie das SDK wie zuvor gezeigt, einschließlich der Installation der erforderlichen Clients und Pakete. Erstellen Sie ein Objekt, das die Parameter zum Festlegen von SMS-Attributen enthält, einschließlich der Namen der einzelnen Attribute, die festgelegt werden, und der jeweils festzulegenden Werte. Einzelheiten zu verfügbaren SMS-Attributen finden Sie unter [Set SMSAttributes](#) in der Amazon Simple Notification Service API-Referenz.

In diesem Beispiel wird das `DefaultSMSType`-Attribut auf `Transactional` festgelegt. Damit wird die Nachrichtenzustellung im Hinblick auf höchste Zuverlässigkeit optimiert. Übergeben Sie die Parameter an die `SetTopicAttributesCommand`-Methode der Client-Klasse SNS. Um die `SetSMSAttributesCommand` Methode aufzurufen, erstellen Sie eine asynchrone Funktion, die ein Amazon SNS SNS-Client-Serviceobjekt aufruft und das Parameter-Objekt übergibt.

```
import { SetSMSAttributesCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {"Transactional" | "Promotional"} defaultSmsType
 */
export const setSmsType = async (defaultSmsType = "Transactional") => {
  const response = await snsClient.send(
    new SetSMSAttributesCommand({
      attributes: {
        // Promotional - (Default) Noncritical messages, such as marketing messages.
        // Transactional - Critical messages that support customer transactions,
        // such as one-time passcodes for multi-factor authentication.
        DefaultSMSType: defaultSmsType,
      },
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '1885b977-2d7e-535e-8214-e44be727e265',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
  return response;
};
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node set-sms-attribute-type.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Überprüfen, ob für eine Telefonnummer der Empfang deaktiviert wurde

In diesem Beispiel verwenden Sie ein Node.js-Modul, um zu überprüfen, ob für eine Telefonnummer der Empfang von SMS-Nachrichten deaktiviert wurde.

Erstellen Sie ein `libs` Verzeichnis und ein Modul Node.js mit dem Dateinamen `snsClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SNS SNS-Client-Objekt erstellt wird. Ersetzen Sie es **REGION** durch Ihre AWS Region.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `check-if-phone-number-is-opted-out.js`. Konfigurieren Sie das SDK wie zuvor dargestellt. Erstellen Sie ein Objekt, das die zu überprüfende Telefonnummer als Parameter enthält.

In diesem Beispiel wird der `PhoneNumber`-Parameter festgelegt, um die Telefonnummer anzugeben, die überprüft werden soll. Übergeben Sie das Objekt an die `CheckIfPhoneNumberIsOptedOutCommand`-Methode der Client-Klasse `SNS`. Um die `CheckIfPhoneNumberIsOptedOutCommand` Methode aufzurufen, erstellen Sie eine asynchrone Funktion, die ein Amazon SNS SNS-Client-Serviceobjekt aufruft und das Parameter-Objekt übergibt.

Note

1.

Durch die **PHONE_NUMBER** Telefonnummer ersetzen.

```
import { CheckIfPhoneNumberIsOptedOutCommand } from "@aws-sdk/client-sns";

import { snsClient } from "../libs/snsClient.js";
```



```
export const checkIfPhoneNumberIsOptedOut = async (
  phoneNumber = "5555555555",
) => {
  const command = new CheckIfPhoneNumberIsOptedOutCommand({
    phoneNumber,
  });

  const response = await snsClient.send(command);
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '3341c28a-cdc8-5b39-a3ee-9fb0ee125732',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   isOptedOut: false
  // }
  return response;
};
```

Um das Beispiel auszuführen, geben Sie in der Befehlszeile Folgendes ein.

```
node check-if-phone-number-is-opted-out.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Auflisten deaktivierter Telefonnummern

In diesem Beispiel verwenden Sie ein Node.js-Modul, um eine Liste der Telefonnummern abzurufen, für die der Empfang von SMS-Nachrichten deaktiviert wurde.

Erstellen Sie ein `libs` Verzeichnis und ein Modul Node.js mit dem Dateinamen `snsClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SNS SNS-Client-Objekt erstellt wird. Ersetzen Sie es **REGION** durch Ihre AWS Region.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
blank
```

```
// the SDK will default to the region set in your AWS config.  
export const snsClient = new SNSClient({});
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `list-phone-numbers-opted-out.js`. Konfigurieren Sie das SDK wie zuvor dargestellt. Erstellen Sie ein leeres Objekt als Parameter.

Übergeben Sie das Objekt an die `ListPhoneNumbersOptedOutCommand`-Methode der Client-Klasse `SNS`. Um die `ListPhoneNumbersOptedOutCommand` Methode aufzurufen, erstellen Sie eine asynchrone Funktion, die ein Amazon SNS `SNS-Client-Service`objekt aufruft und das Parameter-Objekt übergibt.

```
import { ListPhoneNumbersOptedOutCommand } from "@aws-sdk/client-sns";  
import { snsClient } from "../libs/snsClient.js";  
  
export const listPhoneNumbersOptedOut = async () => {  
  const response = await snsClient.send(  
    new ListPhoneNumbersOptedOutCommand({}),  
  );  
  console.log(response);  
  // {  
  //   '$metadata': {  
  //     httpStatusCode: 200,  
  //     requestId: '44ff72fd-1037-5042-ad96-2fc16601df42',  
  //     extendedRequestId: undefined,  
  //     cfId: undefined,  
  //     attempts: 1,  
  //     totalRetryDelay: 0  
  //   },  
  //   phoneNumbers: ['+15555550100']  
  // }  
  return response;  
};
```

Um das Beispiel auszuführen, geben Sie an der Befehlszeile Folgendes ein.

```
node list-phone-numbers-opted-out.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Veröffentlichen einer SMS-Nachricht

In diesem Beispiel verwenden Sie ein Node.js-Modul zum Senden einer SMS-Nachricht an eine Telefonnummer.

Erstellen Sie ein `libs` Verzeichnis und ein Modul Node.js mit dem Dateinamen `snsClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon SNS SNS-Client-Objekt erstellt wird. Ersetzen Sie es **REGION** durch Ihre AWS Region.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `publish-sms.js`. Konfigurieren Sie das SDK wie zuvor gezeigt, einschließlich der Installation der erforderlichen Clients und Pakete. Erstellen Sie ein Objekt, das die Parameter `Message` und `PhoneNumber` enthält.

Wenn Sie eine SMS-Nachricht senden, geben Sie die Telefonnummer im E.164-Format an. Die Richtlinie E.164 legt die internationale Schreibweise für Telefonnummern fest. Rufnummern im E.164-Format bestehen aus maximal 15 Zeichen sowie einem vorangestellten Plus-Zeichen (+) und der Ländervorwahl. Eine US-Telefonnummer im E.164-Format würde beispielsweise als XXX5550100 +1001 angezeigt.

In diesem Beispiel wird der `PhoneNumber`-Parameter festgelegt, um die Telefonnummer zum Senden der Nachricht anzugeben. Übergeben Sie das Objekt an die `PublishCommand`-Methode der Client-Klasse SNS. Um die `PublishCommand` Methode aufzurufen, erstellen Sie eine asynchrone Funktion, die ein Amazon SNS-Serviceobjekt aufruft und das Parameterobjekt übergibt.

Note

TEXT_MESSAGE Ersetzen Sie es durch die Textnachricht und **PHONE_NUMBER** durch die Telefonnummer.

```
import { PublishCommand } from "@aws-sdk/client-sns";
```

```
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string | Record<string, any>} message - The message to send. Can be a plain
 string or an object
 *
 *                                     if you are using the `json`
 `MessageStructure`.
 * @param {*} phoneNumber - The phone number to send the message to.
 */
export const publish = async (
  message = "Hello from SNS!",
  phoneNumber = "+15555555555",
) => {
  const response = await snsClient.send(
    new PublishCommand({
      Message: message,
      // One of PhoneNumber, TopicArn, or TargetArn must be specified.
      PhoneNumber: phoneNumber,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '7410094f-efc7-5f52-af03-54737569ab77',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   MessageId: 'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxxx'
  // }
  return response;
};
```

Um das Beispiel auszuführen, geben Sie in der Befehlszeile Folgendes ein.

```
node publish-sms.js
```

Dieser Beispielcode finden Sie [hier unter GitHub](#).

Beispiele für Amazon Transcribe

Amazon Transcribe macht es Entwicklern leicht, ihren Anwendungen Sprache-zu-Text-Funktionen hinzuzufügen.



Die JavaScript API für Amazon Transcribe wird über die [TranscribeService](#)Client-Klasse verfügbar gemacht.

Themen

- [Beispiele für Amazon Transcribe](#)
- [Medizinische Beispiele von Amazon Transcribe](#)

Beispiele für Amazon Transcribe

In diesem Beispiel werden eine Reihe von Node.js -Modulen verwendet, um Transkriptionsaufträge mit den folgenden Methoden der Client-Klasse zu erstellen, aufzulisten und zu löschen:

TranscribeService

- [StartTranscriptionJobCommand](#)
- [ListTranscriptionJobsCommand](#)
- [DeleteTranscriptionJobCommand](#)

Weitere Informationen zu Amazon Transcribe Transcribe-Benutzern finden Sie im Amazon Transcribe [Transcribe-Entwicklerhandbuch](#).

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Richten Sie die Projektumgebung ein, um diese TypeScript Node-Beispiele auszuführen, und installieren Sie die erforderlichen Module AWS SDK für JavaScript und Module von Drittanbietern. Folgen Sie den Anweisungen auf [GitHub](#).
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zur Bereitstellung einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter [Dateien mit gemeinsam genutzten Konfigurationen und Anmeldeinformationen](#) im AWS SDKs Referenzhandbuch zu Tools.

Important

Diese Beispiele veranschaulichen, wie Sie Objekte und Befehle mithilfe von ECMAScript6 (ES6) als import/export Client verwalten können.

- Dazu ist die Version 13.x von Node.js oder höher erforderlich. Informationen zum Herunterladen und Installieren der neuesten Version von Node.js finden Sie unter [Node.js downloads](#).
- Wenn Sie die CommonJS-Syntax bevorzugen, finden Sie weitere Informationen unter [JavaScript ES6/CommonJs-Syntax](#)

Einen Amazon Transcribe Transcribe-Job starten

Dieses Beispiel zeigt, wie Sie einen Amazon Transcribe-Transkriptionsauftrag mit dem starten. AWS SDK für JavaScript Weitere Informationen finden Sie unter [StartTranscriptionJobCommand](#).

Erstellen Sie ein `libs` Verzeichnis und ein Modul `Node.js` mit dem Dateinamen.

`transcribeClient.js` Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon Transcribe-Client-Objekt erstellt wird. Ersetzen Sie es **REGION** durch Ihre Region AWS .

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `transcribe-create-job.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor gezeigt konfigurieren, einschließlich der Installation der erforderlichen Clients und Pakete. Erstellen Sie ein Parameterobjekt und geben Sie die erforderlichen Parameter an. Starten Sie den Job mit dem `StartMedicalTranscriptionJobCommand` Befehl.

Note

MEDICAL_JOB_NAME Ersetzen Sie ihn durch einen Namen für den Transkriptionsjob.
OUTPUT_BUCKET_NAME Geben Sie für den Amazon S3 S3-Bucket an, in dem die Ausgabe gespeichert wird. Zur ***JOB_TYPE*** Angabe von Auftragsstypen. ***SOURCE_LOCATION*** Geben Sie dazu den Speicherort der Quelldatei an. ***SOURCE_FILE_LOCATION*** Geben Sie dazu den Speicherort der Eingabemediendatei an.

```
// Import the required AWS SDK clients and commands for Node.js
import { StartTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "../libs/transcribeClient.js";

// Set the parameters
export const params = {
  TranscriptionJobName: "JOB_NAME",
  LanguageCode: "LANGUAGE_CODE", // For example, 'en-US'
  MediaFormat: "SOURCE_FILE_FORMAT", // For example, 'wav'
  Media: {
    MediaFileUri: "SOURCE_LOCATION",
    // For example, "https://transcribe-demo.s3-REGION.amazonaws.com/hello_world.wav"
  },
  OutputBucketName: "OUTPUT_BUCKET_NAME",
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new StartTranscriptionJobCommand(params),
    );
    console.log("Success - put", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
```

```
run();
```

Um das Beispiel auszuführen, geben Sie in der Befehlszeile Folgendes ein.

```
node transcribe-create-job.js
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Amazon Transcribe Transcribe-Jobs auflisten

Dieses Beispiel zeigt, wie die Amazon Transcribe-Transkriptionsaufträge mithilfe von aufgelistet werden. AWS SDK für JavaScript Weitere Informationen darüber, welche anderen Einstellungen Sie ändern können, finden Sie unter [ListTranscriptionJobCommand](#)

Erstellen Sie ein `libs` Verzeichnis und ein Modul `Node.js` mit dem Dateinamen `transcribeClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon Transcribe-Client-Objekt erstellt wird. Ersetzen Sie es **REGION** durch Ihre Region AWS .

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein `Node.js`-Modul mit dem Dateinamen `transcribe-list-jobs.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor gezeigt konfigurieren, einschließlich der Installation der erforderlichen Clients und Pakete. Erstellen Sie ein Parameterobjekt mit den erforderlichen Parametern.

Note

KEY_WORD Ersetzen Sie es durch ein Schlüsselwort, das der zurückgegebene Jobname enthalten muss.


```
// Import the required AWS SDK clients and commands for Node.js

import { ListTranscriptionJobsCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "../libs/transcribeClient.js";

// Set the parameters
export const params = {
  JobNameContains: "KEYWORD", // Not required. Returns only transcription
  // job names containing this string
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new ListTranscriptionJobsCommand(params),
    );
    console.log("Success", data.TranscriptionJobSummaries);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};

run();
```

Um das Beispiel auszuführen, geben Sie in der Befehlszeile Folgendes ein.

```
node transcribe-list-jobs.js
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Löschen eines Amazon Transcribe Transcribe-Jobs

Dieses Beispiel zeigt, wie Sie einen Amazon Transcribe-Transkriptionsauftrag mit dem löschen. AWS SDK für JavaScript Weitere Informationen zu optionalen Optionen finden Sie unter.

[DeleteTranscriptionJobCommand](#)

Erstellen Sie ein `libs` Verzeichnis und ein Modul `Node.js` mit dem Dateinamen `transcribeClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon Transcribe-Client-Objekt erstellt wird. Ersetzen Sie es **REGION** durch Ihre Region AWS .

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
```

```
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create Transcribe service object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein Node.js-Modul mit dem Dateinamen `transcribe-delete-job.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor gezeigt konfigurieren, einschließlich der Installation der erforderlichen Clients und Pakete. Geben Sie die AWS Region und den Namen des Jobs an, den Sie löschen möchten.

 Note

JOB_NAME Ersetzen Sie ihn durch den Namen des zu löschenden Jobs.

```
// Import the required AWS SDK clients and commands for Node.js
import { DeleteTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "../libs/transcribeClient.js";

// Set the parameters
export const params = {
  TranscriptionJobName: "JOB_NAME", // Required. For example, 'transcription_demo'
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new DeleteTranscriptionJobCommand(params),
    );
    console.log("Success - deleted");
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Um das Beispiel auszuführen, geben Sie in der Befehlszeile Folgendes ein.

```
node transcribe-delete-job.js
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Medizinische Beispiele von Amazon Transcribe

In diesem Beispiel werden eine Reihe von Modulen von Node.js verwendet, um medizinische Transkriptionsaufträge mit den folgenden Methoden der Client-Klasse zu erstellen, aufzulisten und zu löschen: `TranscribeService`

- [StartMedicalTranscriptionJobCommand](#)
- [ListMedicalTranscriptionJobsCommand](#)
- [DeleteMedicalTranscriptionJobCommand](#)

Weitere Informationen zu Amazon Transcribe Transcribe-Benutzern finden Sie im [Amazon Transcribe Transcribe-Entwicklerhandbuch](#).

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Richten Sie die Projektumgebung ein, um diese TypeScript Node-Beispiele auszuführen, und installieren Sie die erforderlichen Module AWS SDK für JavaScript und Module von Drittanbietern. Folgen Sie den Anweisungen auf [GitHub](#).
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zur Bereitstellung einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter [Dateien mit gemeinsam genutzten Konfigurationen und Anmeldeinformationen](#) im AWS SDKs Referenzhandbuch zu Tools.

Important

Diese Beispiele veranschaulichen, wie Sie Objekte und Befehle mithilfe von ECMAScript6 (ES6) als import/export Client verwalten können.

- Dazu ist die Version 13.x von Node.js oder höher erforderlich. Informationen zum Herunterladen und Installieren der neuesten Version von Node.js finden Sie unter [Node.js downloads](#).

- Wenn Sie die CommonJS-Syntax bevorzugen, finden Sie weitere Informationen unter [JavaScript ES6/CommonJs-Syntax](#)

Einen medizinischen Transkriptionsauftrag mit Amazon Transcribe starten

Dieses Beispiel zeigt, wie Sie einen medizinischen Transkriptionsauftrag von Amazon Transcribe mit dem `start` starten. AWS SDK für JavaScript Weitere Informationen finden Sie unter [startMedicalTranscriptionJob](#).

Erstellen Sie ein `libs` Verzeichnis und ein Modul `Node.js` mit dem Dateinamen `transcribeClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon Transcribe-Client-Objekt erstellt wird. Ersetzen Sie es **REGION** durch Ihre Region AWS .

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create Transcribe service object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein `Node.js`-Modul mit dem Dateinamen `transcribe-create-medical-job.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor gezeigt konfigurieren, einschließlich der Installation der erforderlichen Clients und Pakete. Erstellen Sie ein Parameterobjekt und geben Sie die erforderlichen Parameter an. Starten Sie den medizinischen Job mit dem `StartMedicalTranscriptionJobCommand` Befehl.

Note

MEDICAL_JOB_NAME Ersetzen Sie ihn durch einen Namen für den medizinischen Transkriptionsjob. **OUTPUT_BUCKET_NAME** Geben Sie für den Amazon S3 S3-Bucket an, in dem die Ausgabe gespeichert wird. Zur **JOB_TYPE** Angabe von Auftragsstypen. **SOURCE_LOCATION** Geben Sie dazu den Speicherort der Quelldatei an. **SOURCE_FILE_LOCATION** Geben Sie dazu den Speicherort der Eingabemediendatei an.

```
// Import the required AWS SDK clients and commands for Node.js
import { StartMedicalTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "../libs/transcribeClient.js";

// Set the parameters
export const params = {
  MedicalTranscriptionJobName: "MEDICAL_JOB_NAME", // Required
  OutputBucketName: "OUTPUT_BUCKET_NAME", // Required
  Specialty: "PRIMARYCARE", // Required. Possible values are 'PRIMARYCARE'
  Type: "JOB_TYPE", // Required. Possible values are 'CONVERSATION' and 'DICTATION'
  LanguageCode: "LANGUAGE_CODE", // For example, 'en-US'
  MediaFormat: "SOURCE_FILE_FORMAT", // For example, 'wav'
  Media: {
    MediaFileUri: "SOURCE_FILE_LOCATION",
    // The S3 object location of the input media file. The URI must be in the same
    // region
    // as the API endpoint that you are calling. For example,
    // "https://transcribe-demo.s3-REGION.amazonaws.com/hello_world.wav"
  },
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new StartMedicalTranscriptionJobCommand(params),
    );
    console.log("Success - put", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};

run();
```

Um das Beispiel auszuführen, geben Sie in der Befehlszeile Folgendes ein.

```
node transcribe-create-medical-job.js
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Stellenangebote von Amazon Transcribe im Gesundheitswesen auflisten

Dieses Beispiel zeigt, wie Sie die Amazon Transcribe-Transkriptionsaufträge mithilfe von auflisten. AWS SDK für JavaScript Weitere Informationen finden Sie unter [ListTranscriptionMedicalJobsCommand](#).

Erstellen Sie ein `libs` Verzeichnis und ein Modul `Node.js` mit dem Dateinamen `transcribeClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon Transcribe-Client-Objekt erstellt wird. Ersetzen Sie es **REGION** durch Ihre Region AWS .

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein `Node.js`-Modul mit dem Dateinamen `transcribe-list-medical-jobs.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor gezeigt konfigurieren, einschließlich der Installation der erforderlichen Clients und Pakete. Erstellen Sie ein Parameterobjekt mit den erforderlichen Parametern und listen Sie die medizinischen Aufgaben mithilfe des `ListMedicalTranscriptionJobsCommand` Befehls auf.

Note

KEYWORD Ersetzen Sie es durch ein Schlüsselwort, das der zurückgegebene Jobname enthalten muss.

```
// Import the required AWS SDK clients and commands for Node.js

import { ListMedicalTranscriptionJobsCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "../libs/transcribeClient.js";

// Set the parameters
export const params = {
  JobNameContains: "KEYWORD", // Returns only transcription job names containing this
  string
};
```

```
export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new ListMedicalTranscriptionJobsCommand(params),
    );
    console.log("Success", data.MedicalTranscriptionJobName);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Um das Beispiel auszuführen, geben Sie in der Befehlszeile Folgendes ein.

```
node transcribe-list-medical-jobs.js
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Löschen eines medizinischen Jobs bei Amazon Transcribe

Dieses Beispiel zeigt, wie Sie einen Amazon Transcribe-Transkriptionsauftrag mit dem löschen. AWS SDK für JavaScript Weitere Informationen zu optionalen Optionen finden Sie unter.

[DeleteTranscriptionMedicalJobCommand](#)

Erstellen Sie ein `libs` Verzeichnis und ein Modul `Node.js` mit dem Dateinamen `transcribeClient.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein, wodurch das Amazon Transcribe-Client-Objekt erstellt wird. Ersetzen Sie es **REGION** durch Ihre Region AWS .

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create Transcribe service object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Erstellen Sie ein `Node.js`-Modul mit dem Dateinamen `transcribe-delete-job.js`. Stellen Sie sicher, dass Sie das SDK wie zuvor gezeigt konfigurieren, einschließlich der Installation

der erforderlichen Clients und Pakete. Erstellen Sie ein Parameterobjekt mit den erforderlichen Parametern und löschen Sie den medizinischen Job mithilfe des `DeleteMedicalJobCommand` Befehls.

 Note

JOB_NAME Ersetzen Sie es durch den Namen des zu löschenden Jobs.

```
// Import the required AWS SDK clients and commands for Node.js
import { DeleteMedicalTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "../libs/transcribeClient.js";

// Set the parameters
export const params = {
  MedicalTranscriptionJobName: "MEDICAL_JOB_NAME", // For example,
  'medical_transcription_demo'
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new DeleteMedicalTranscriptionJobCommand(params),
    );
    console.log("Success - deleted");
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};

run();
```

Um das Beispiel auszuführen, geben Sie in der Befehlszeile Folgendes ein.

```
node transcribe-delete-medical-job.js
```

Diesen Beispielcode finden Sie [hier auf GitHub](#).

Node.js auf einer Amazon EC2 EC2-Instance einrichten

Ein gängiges Szenario für die Verwendung von Node.js mit dem SDK für JavaScript ist die Einrichtung und Ausführung einer Node.js -Webanwendung auf einer Amazon Elastic Compute Cloud (Amazon EC2) -Instance. In diesem Tutorial erstellen Sie eine Linux-Instance, stellen eine Verbindung zur Instance über SSH her und installieren anschließend Node.js, um es auf dieser Instance auszuführen.

Voraussetzungen

In diesem Tutorial wird davon ausgegangen, dass Sie bereits eine Linux-Instance mit einem öffentlichen DNS-Namen gestartet haben, der über das Internet erreichbar ist und zu der Sie über SSH eine Verbindung herstellen können. Weitere Informationen finden Sie unter [Schritt 1: Starten einer Instance](#) im Benutzerhandbuch zu Amazon EC2.

Important

Verwenden Sie das Amazon Linux 2023 Amazon Machine Image (AMI), wenn Sie eine neue Amazon EC2 EC2-Instance starten.

Außerdem müssen Sie Ihre Sicherheitsgruppe so konfiguriert haben, dass Verbindungen über SSH (Port 22), HTTP (Port 80) und HTTPS (Port 443) erlaubt sind. Weitere Informationen zu diesen Voraussetzungen finden Sie unter [Einrichtung mit Amazon EC2](#) im Amazon EC2 EC2-Benutzerhandbuch.

Verfahren

Mithilfe des folgenden Verfahrens können Sie Node.js auf einer Amazon Linux-Instance installieren. Sie können diesen Server zum Hosten einer Node.js-Webanwendung verwenden.

So richten Sie Node.js auf Ihrer Linux-Instance ein

1. Stellen Sie als `ec2-user` eine Verbindung mit Ihrer Linux-Instance über SSH her.
2. Installieren Sie den Node Version Manager (`nvm`), indem Sie in der Befehlszeile Folgendes eingeben.

 Warning

AWS steuert den folgenden Code nicht. Bevor Sie ihn ausführen, überprüfen Sie unbedingt dessen Authentizität und Integrität. Weitere Informationen zu diesem Code finden Sie im [GitHubnvm-Repository](#).

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.7/install.sh | bash
```

Wir werden Node.js verwenden, da mehrere Versionen von Node.js installiert werden können und Sie zwischen ihnen wechseln können.

3. Laden Sie, nvm indem Sie in der Befehlszeile Folgendes eingeben.

```
source ~/.bashrc
```

4. Verwenden Sie nvm, um die neueste LTS-Version von Node.js zu installieren, indem Sie in der Befehlszeile Folgendes eingeben.

```
nvm install --lts
```

Bei der Installation von Node.js wird auch der Node Package Manager (npm) installiert, sodass Sie bei Bedarf zusätzliche Module installieren können.

5. Testen Sie, ob Node.js installiert ist und ordnungsgemäß ausgeführt wird. Geben Sie dazu den folgenden Befehl in die Befehlszeile ein.

```
node -e "console.log('Running Node.js ' + process.version)"
```

Dadurch erscheint folgende Meldung, in der die ausgeführte Node.js-Version angezeigt wird.

Running Node.js **VERSION**

 Note

Die Knoteninstallation gilt nur für die aktuelle Amazon EC2-Sitzung. Wenn Sie Ihre CLI-Sitzung neu starten, müssen Sie nvm erneut verwenden, um die installierte Knotenversion zu aktivieren. Wenn die Instance beendet wird, müssen Sie den Knoten erneut installieren. Die

Alternative besteht darin, ein Amazon Machine Image (AMI) der Amazon EC2 EC2-Instance zu erstellen, sobald Sie die Konfiguration haben, die Sie behalten möchten, wie im folgenden Thema beschrieben.

Ein Amazon Machine Image (AMI) erstellen

Nachdem Sie Node.js auf einer Amazon EC2 EC2-Instance installiert haben, können Sie aus dieser Instance ein Amazon Machine Image (AMI) erstellen. Die Erstellung eines AMI macht es einfach, mehrere Amazon EC2 EC2-Instances mit derselben Node.js Installation bereitzustellen. Weitere Informationen zum Erstellen eines AMI aus einer vorhandenen Instance finden Sie unter [Creating an amazon EBS-backed Linux AMI](#) im Amazon EC2 EC2-Benutzerhandbuch.

Zugehörige Ressourcen

Weitere Informationen zu den in diesem Thema verwendeten Befehlen und der Software finden Sie auf den folgenden Webseiten:

- Node Version Manager (nvm) — Siehe [nvm](#) repo on. GitHub
- Node Package Manager (npm) — Siehe [npm-Website](#).

Lambda mit API Gateway aufrufen

Sie können eine Lambda-Funktion mithilfe von Amazon API Gateway aufrufen. Dabei handelt es sich um einen AWS Service zum Erstellen, Veröffentlichen, Verwalten, Überwachen und Sichern von REST, HTTP und WebSocket APIs in großem Maßstab. API-Entwickler können APIs diesen Zugriff AWS oder andere Webdienste sowie in der AWS Cloud gespeicherte Daten erstellen. Als API Gateway Gateway-Entwickler können Sie Anwendungen APIs für die Verwendung in Ihren eigenen Client-Anwendungen erstellen. Weitere Informationen finden Sie unter [Was ist Amazon API Gateway](#).

AWS Lambda ist ein Rechenservice, mit dem Sie Code ausführen können, ohne Server bereitzustellen oder zu verwalten. Sie können Lambda-Funktionen in verschiedenen Programmiersprachen erstellen. Weitere Informationen zu finden Sie AWS Lambda unter [Was ist AWS Lambda](#).

In diesem Beispiel erstellen Sie eine Lambda-Funktion mithilfe der JavaScript Lambda-Laufzeit-API. In diesem Beispiel werden verschiedene AWS Dienste aufgerufen, um einen bestimmten Anwendungsfall auszuführen. Nehmen wir beispielsweise an, dass eine Organisation ihren

Mitarbeitern eine mobile Textnachricht sendet, in der sie zum einjährigen Jubiläum gratuliert, wie in dieser Abbildung gezeigt.



Die Fertigstellung des Beispiels sollte etwa 20 Minuten in Anspruch nehmen.

Dieses Beispiel zeigt Ihnen, wie Sie mithilfe von JavaScript Logik eine Lösung erstellen, die diesen Anwendungsfall erfüllt. Sie lernen beispielsweise, eine Datenbank zu lesen, um festzustellen, welche Mitarbeiter das einjährige Jubiläum erreicht haben, wie die Daten verarbeitet und eine Textnachricht mithilfe einer Lambda-Funktion versendet werden. Anschließend erfahren Sie, wie Sie API Gateway verwenden, um diese AWS Lambda Funktion mithilfe eines Rest-Endpunkts aufzurufen. Sie können die Lambda-Funktion beispielsweise mit dem folgenden curl-Befehl aufrufen:

```
curl -XGET "https://xxxxqjko1o3.execute-api.us-east-1.amazonaws.com/cronstage/employee"
```

AWS In diesem Tutorial wird eine Amazon DynamoDB-Tabelle mit dem Namen Employee verwendet, die diese Felder enthält.

- id — der Primärschlüssel für die Tabelle.
- FirstName — Vorname des Mitarbeiters.
- Telefon — Telefonnummer des Mitarbeiters.
- StartDate — Startdatum des Mitarbeiters.

Scan

[Table] Employee: Id

+ Add filter

Start search

	Id	first	phone	startDate
☐	1	Scott	15555555654	2019-12-20
☐	2	Malcolm	15555555654	2019-12-17
☐	55	Lam	15555555654	2019-12-19

⚠ Important

Bearbeitungskosten: Die in diesem Dokument enthaltenen AWS Dienste sind im AWS kostenlosen Kontingent enthalten. Achten Sie jedoch darauf, alle Ressourcen zu beenden, nachdem Sie dieses Beispiel abgeschlossen haben, um sicherzustellen, dass Ihnen nichts in Rechnung gestellt wird.

Um die App zu erstellen:

1. [Vollständige Voraussetzungen](#)
2. [Erstellen Sie die AWS Ressourcen](#)
3. [Bereiten Sie das Browser-Skript vor](#)
4. [Lambda-Funktion erstellen und hochladen](#)
5. [Stellen Sie die Lambda-Funktion bereit](#)
6. [Führen Sie die App aus](#)
7. [Löschen Sie die Ressourcen](#)

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Richten Sie die Projektumgebung ein, um diese TypeScript Node-Beispiele auszuführen, und installieren Sie die erforderlichen Module AWS SDK für JavaScript und Module von Drittanbietern. Folgen Sie den Anweisungen auf [GitHub](#).

- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zur Bereitstellung einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter [Dateien mit gemeinsam genutzten Konfigurationen und Anmeldeinformationen](#) im AWS SDKs Referenzhandbuch zu Tools.

Erstellen Sie die AWS Ressourcen

Für dieses Tutorial sind die folgenden Ressourcen erforderlich:

- Eine Amazon DynamoDB-Tabelle, benannt Employee mit einem Schlüssel Id und den Feldern, die in der vorherigen Abbildung gezeigt wurden. Stellen Sie sicher, dass Sie die richtigen Daten eingeben, einschließlich eines gültigen Mobiltelefons, mit dem Sie diesen Anwendungsfall testen möchten. Weitere Informationen finden Sie unter [Tabelle erstellen](#).
- Eine IAM-Rolle mit angehängten Berechtigungen zur Ausführung von Lambda-Funktionen.
- Ein Amazon S3 S3-Bucket zum Hosten der Lambda-Funktion.

Sie können diese Ressourcen manuell erstellen, wir empfehlen jedoch, diese Ressourcen CloudFormation wie in diesem Tutorial beschrieben bereitzustellen.

Erstellen Sie die AWS Ressourcen mit CloudFormation

CloudFormation ermöglicht es Ihnen, AWS Infrastrukturbereitstellungen vorhersehbar und wiederholt zu erstellen und bereitzustellen. [Weitere Informationen zu CloudFormation finden Sie im AWS CloudFormation Benutzerhandbuch](#).

Um den CloudFormation Stack zu erstellen, verwenden Sie AWS CLI:

1. Installieren und konfigurieren Sie die AWS CLI folgenden Anweisungen im [AWS CLI Benutzerhandbuch](#).
2. Erstellen Sie eine Datei mit dem Namen `setup.yaml` im Stammverzeichnis Ihres Projektordners und kopieren Sie den Inhalt [hier GitHub](#) hinein.

Note

Die CloudFormation Vorlage wurde unter Verwendung der [hier AWS CDK verfügbaren Datei](#) generiert GitHub. Weitere Informationen zu finden Sie im [AWS Cloud Development Kit \(AWS CDK\) Entwicklerhandbuch](#). AWS CDK

3. Führen Sie den folgenden Befehl von der Befehlszeile aus und **STACK_NAME** ersetzen Sie ihn durch einen eindeutigen Namen für den Stack.

⚠ Important

Der Stack-Name muss innerhalb einer AWS Region und eines AWS Kontos eindeutig sein. Sie können bis zu 128 Zeichen angeben. Zahlen und Bindestriche sind zulässig.

```
aws cloudformation create-stack --stack-name STACK_NAME --template-body file://  
setup.yaml --capabilities CAPABILITY_IAM
```

Weitere Informationen zu den `create-stack` Befehlsparametern finden Sie in der [AWS CLI Befehlsreferenz](#) und im [CloudFormation Benutzerhandbuch](#).

4. Füllen Sie als Nächstes die Tabelle wie folgt [Füllen der Tabelle](#) aus.

Füllen der Tabelle

Um die Tabelle zu füllen, erstellen Sie zunächst ein Verzeichnis mit dem Namen `libs` und darin eine Datei mit dem Namen `DynamoClient.js` , und fügen Sie den folgenden Inhalt ein.

```
const { DynamoDBClient } = require ( "@aws-sdk/client-dynamodb" );  
// Set the AWS Region.  
const REGION = "REGION"; // e.g. "us-east-1"  
// Create an Amazon Lambda service client object.  
const dynamoClient = new DynamoDBClient({region:REGION});  
module.exports = { dynamoClient };
```

Dieser Code ist [hier verfügbar GitHub](#).

Erstellen Sie als Nächstes eine Datei mit dem Namen `populate-table.js` im Stammverzeichnis Ihres Projektordners und kopieren Sie den Inhalt [hier GitHub](#) hinein. Ersetzen Sie für eines der Elemente den Wert für die `phone` Eigenschaft durch eine gültige Handynummer im E.164-Format und den Wert für `startDate` durch das heutige Datum.

Führen Sie den folgenden Befehl von der Befehlszeile aus.

```
node populate-table.js
```

```
const { BatchWriteItemCommand } = require ( "aws-sdk/client-dynamodb" );
const {dynamoClient} = require ( "../libs/dynamoClient" );

// Set the parameters.
export const params = {
  RequestItems: {
    Employees: [
      {
        PutRequest: {
          Item: {
            id: { N: "1" },
            firstName: { S: "Bob" },
            phone: { N: "1555555555555654" },
            startDate: { S: "2019-12-20" },
          },
        },
      },
      {
        PutRequest: {
          Item: {
            id: { N: "2" },
            firstName: { S: "Xing" },
            phone: { N: "1555555555555653" },
            startDate: { S: "2019-12-17" },
          },
        },
      },
      {
        PutRequest: {
          Item: {
            id: { N: "55" },
            firstName: { S: "Harriette" },
            phone: { N: "1555555555555652" },
            startDate: { S: "2019-12-19" },
          },
        },
      },
    ],
  },
};
```



```
export const run = async () => {
  try {
    const data = await dbclient.send(new BatchWriteItemCommand(params));
    console.log("Success", data);
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Dieser Code ist [hier auf](#) verfügbar GitHub.

Die AWS Lambda Funktion erstellen

Konfigurieren des SDKs

Erstellen Sie im `libs` Verzeichnis Dateien mit dem Namen `snsClient.js` und `lambdaClient.js` und fügen Sie den folgenden Inhalt jeweils in diese Dateien ein.

```
const { SNSClient } = require("@aws-sdk/client-sns");
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon SNS service client object.
const snsClient = new SNSClient({ region: REGION });
module.exports = { snsClient };
```

REGIONDurch die AWS Region ersetzen. Dieser Code ist [hier verfügbar GitHub](#).

```
const { LambdaClient } = require("@aws-sdk/client-lambda");
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Lambda service client object.
const lambdaClient = new LambdaClient({ region: REGION });
module.exports = { lambdaClient };
```

REGIONDurch die AWS Region ersetzen. Dieser Code ist [hier verfügbar GitHub](#).

Importieren Sie zunächst die erforderlichen AWS SDK für JavaScript (v3) Module und Befehle. Dann berechne das heutige Datum und weise es einem Parameter zu. Drittens erstellen Sie die Parameter

für den `ScanCommand`. `TABLE_NAME` Ersetzen Sie sie durch den Namen der Tabelle, die Sie im [Erstellen Sie die AWS Ressourcen](#) Abschnitt dieses Beispiels erstellt haben.

Der folgende Codeausschnitt veranschaulicht diesen Schritt. (Das vollständige Beispiel finden Sie unter [Bündelung der Lambda-Funktion](#).)

```
const { ScanCommand } = require("@aws-sdk/client-dynamodb");
const { PublishCommand } = require("@aws-sdk/client-sns");
const { snsClient } = require("../libs/snsClient");
const { dynamoClient } = require("../libs/dynamoClient");

// Get today's date.
const today = new Date();
const dd = String(today.getDate()).padStart(2, "0");
const mm = String(today.getMonth() + 1).padStart(2, "0"); //January is 0!
const yyyy = today.getFullYear();
const date = `${yyyy}-${mm}-${dd}`;

// Set the parameters for the ScanCommand method.
const params = {
  // Specify which items in the results are returned.
  FilterExpression: "startDate = :topic",
  // Define the expression attribute value, which are substitutes for the values you
  want to compare.
  ExpressionAttributeValues: {
    ":topic": { S: date },
  },
  // Set the projection expression, which are the attributes that you want.
  ProjectionExpression: "firstName, phone",
  TableName: "Employees",
};
```

Scannen der DynamoDB-Tabelle

Erstellen Sie zunächst eine `async/await` Funktion, die aufgerufen wird `sendText`, um eine Textnachricht mithilfe von Amazon SNS `PublishCommand` zu veröffentlichen. Fügen Sie dann ein `try` Blockmuster hinzu, das die DynamoDB-Tabelle nach Mitarbeitern durchsucht, deren heutiges Arbeitsjubiläum ansteht. Anschließend wird die `sendText` Funktion aufgerufen, diesen Mitarbeitern eine Textnachricht zu senden. Wenn ein Fehler auftritt, wird der `catch` Block aufgerufen.

Der folgende Codeausschnitt veranschaulicht diesen Schritt. (Das vollständige Beispiel finden Sie unter [Bündelung der Lambda-Funktion](#).)

```
// Helper function to send message using Amazon SNS.
exports.handler = async () => {
  // Helper function to send message using Amazon SNS.
  async function sendText(textParams) {
    try {
      await snsClient.send(new PublishCommand(textParams));
      console.log("Message sent");
    } catch (err) {
      console.log("Error, message not sent ", err);
    }
  }
  try {
    // Scan the table to identify employees with work anniversary today.
    const data = await dynamoClient.send(new ScanCommand(params));
    for (const element of data.Items) {
      const textParams = {
        PhoneNumber: element.phone.N,
        Message: `Hi ${element.firstName.S}; congratulations on your work anniversary!`,
      };
      // Send message using Amazon SNS.
      sendText(textParams);
    }
  } catch (err) {
    console.log("Error, could not scan table ", err);
  }
};
```

Bündelung der Lambda-Funktion

In diesem Thema wird beschrieben, wie Sie die `mylambdafunction.ts` und die erforderlichen AWS SDK für JavaScript Module für dieses Beispiel in einer gebündelten Datei namens `bündeln.index.js`

1. Falls Sie es noch nicht getan haben, folgen Sie den Anweisungen [Erforderliche Aufgaben](#) für dieses Beispiel, um Webpack zu installieren.

Note

Informationen zu Webpack finden Sie unter. [Bündeln Sie Anwendungen mit Webpack](#)

2. Führen Sie in der Befehlszeile den folgenden Befehl aus, um das JavaScript für dieses Beispiel in einer Datei namens `<index.js>` zu bündeln:

```
webpack mylambdafunction.ts --mode development --target node --devtool false --
output-library-target umd -o index.js
```

Important

Beachten Sie, dass die Ausgabe benannt ist `index.js`. Dies liegt daran, dass Lambda-Funktionen einen `index.js` Handler haben müssen, um zu funktionieren.

3. Komprimieren Sie die gebündelte Ausgabedatei, `index.js`, in eine ZIP-Datei mit dem Namen `mylambdafunction.zip`
4. Laden Sie `mylambdafunction.zip` es in den Amazon S3 S3-Bucket hoch, den Sie im [Erstellen Sie die AWS Ressourcen](#) Thema dieses Tutorials erstellt haben.

Stellen Sie die Lambda-Funktion bereit

Erstellen Sie im Stammverzeichnis Ihres Projekts eine `lambda-function-setup.ts` Datei und fügen Sie den folgenden Inhalt ein.

BUCKET_NAME Ersetzen Sie es durch den Namen des Amazon S3 S3-Buckets, in den Sie die ZIP-Version Ihrer Lambda-Funktion hochgeladen haben. **ZIP_FILE_NAME** Ersetzen Sie die ZIP-Version Ihrer Lambda-Funktion durch den Namen oder Namen. **ROLE** Ersetzen Sie es durch die Amazon-Ressourcennummer (ARN) der IAM-Rolle, die Sie im [Erstellen Sie die AWS Ressourcen](#) Thema dieses Tutorials erstellt haben. **LAMBDA_FUNCTION_NAME** Ersetzen Sie durch einen Namen für die Lambda-Funktion.

```
// Load the required Lambda client and commands.
const {
  CreateFunctionCommand
} = require ( "@aws-sdk/client-lambda" );
const { lambdaClient } = require ( "../libs/lambdaClient.js" );

// Set the parameters.
const params = {
  Code: {
    S3Bucket: "BUCKET_NAME", // BUCKET_NAME
    S3Key: "ZIP_FILE_NAME", // ZIP_FILE_NAME
```

```
    },
    FunctionName: "LAMBDA_FUNCTION_NAME",
    Handler: "index.handler",
    Role: "IAM_ROLE_ARN", // IAM_ROLE_ARN; e.g., arn:aws:iam::650138640062:role/v3-
    lambda-tutorial-lambda-role
    Runtime: "nodejs12.x",
    Description:
      "Scans a DynamoDB table of employee details and using Amazon Simple Notification
      Services (Amazon SNS) to " +
      "send employees an email on each anniversary of their start-date.",
  };

const run = async () => {
  try {
    const data = await lambdaClient.send(new CreateFunctionCommand(params));
    console.log("Success", data); // successful response
  } catch (err) {
    console.log("Error", err); // an error occurred
  }
};

run();
```

Geben Sie in der Befehlszeile Folgendes ein, um die Lambda-Funktion bereitzustellen.

```
node lambda-function-setup.ts
```

Dieses Codebeispiel ist [hier verfügbar GitHub](#).

API Gateway so konfigurieren, dass die Lambda-Funktion aufgerufen wird

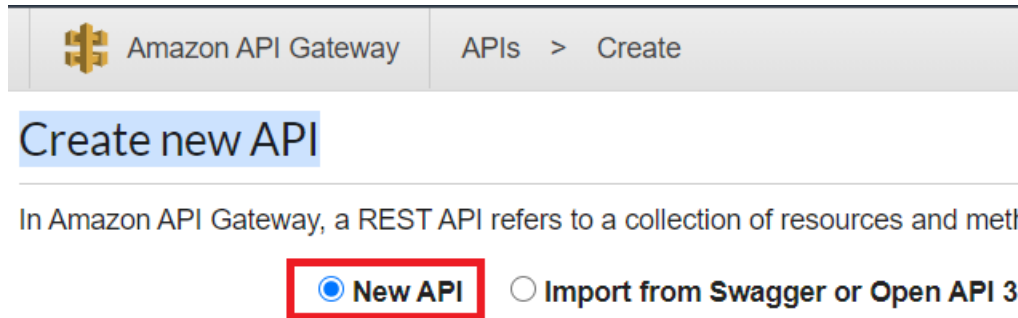
Um die App zu erstellen:

1. [Erstellen Sie die Rest-API](#)
2. [Testen Sie die API Gateway Gateway-Methode](#)
3. [Stellen Sie die API Gateway Gateway-Methode bereit](#)

Erstellen Sie die Rest-API

Sie können die API Gateway Gateway-Konsole verwenden, um einen Rest-Endpunkt für die Lambda-Funktion zu erstellen. Sobald Sie fertig sind, können Sie die Lambda-Funktion mit einem RESTful-Aufruf aufrufen.

1. Melden Sie sich bei der [Amazon API Gateway Gateway-Konsole](#) an.
2. Wählen Sie unter Rest API die Option Build aus.
3. Wählen Sie Neue API aus.



Amazon API Gateway APIs > Create

Create new API

In Amazon API Gateway, a REST API refers to a collection of resources and methods.

☒ **New API** ☐ Import from Swagger or Open API 3

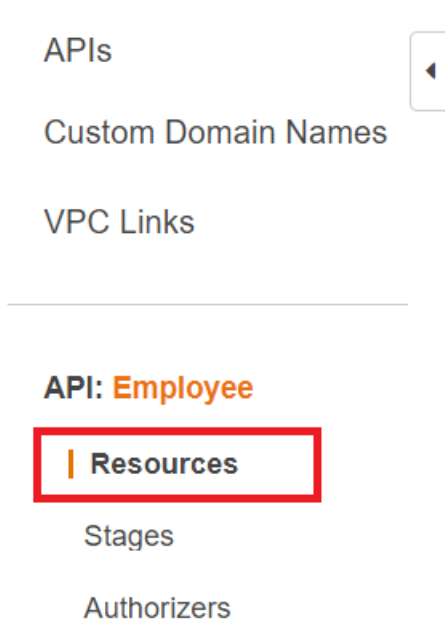
4. Geben Sie Employee als API-Namen an und geben Sie eine Beschreibung ein.

Settings

Choose a friendly name and description for your API.

API name*	Employee
Description	This invokes a Lambda function
Endpoint Type	Regional ⓘ

5. Wählen Sie Create API (API erstellen) aus.
6. Wählen Sie im Bereich Mitarbeiter die Option Ressourcen aus.



7. Geben Sie im Namensfeld Mitarbeiter an.
8. Wählen Sie Create Resources (Ressourcen erstellen) aus.
9. Wählen Sie im Drop-down-Menü Aktionen die Option Ressourcen erstellen aus.

Use this page to create a new child resource for your resource. 🟡

Configure as [proxy resource](#)

☐ ⓘ

Resource Name*

employees

Resource Path*

/ employees

You can add path parameters using brackets. For example, the resource path `{username}` represents a path parameter called 'username'. Configuring `/{proxy+}` as a proxy resource catches all requests to its sub-resources. For example, it works for a GET request to `/foo`. To handle requests to `/`, add a new ANY method on the `/` resource.

Enable API Gateway CORS

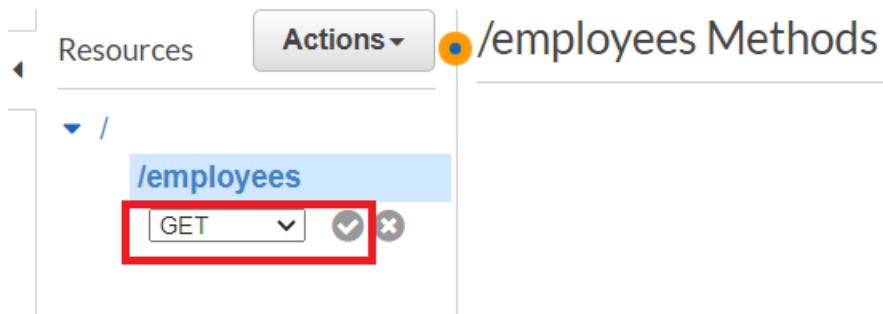
☐ ⓘ

* Required

[Cancel](#)

[Create Resource](#)

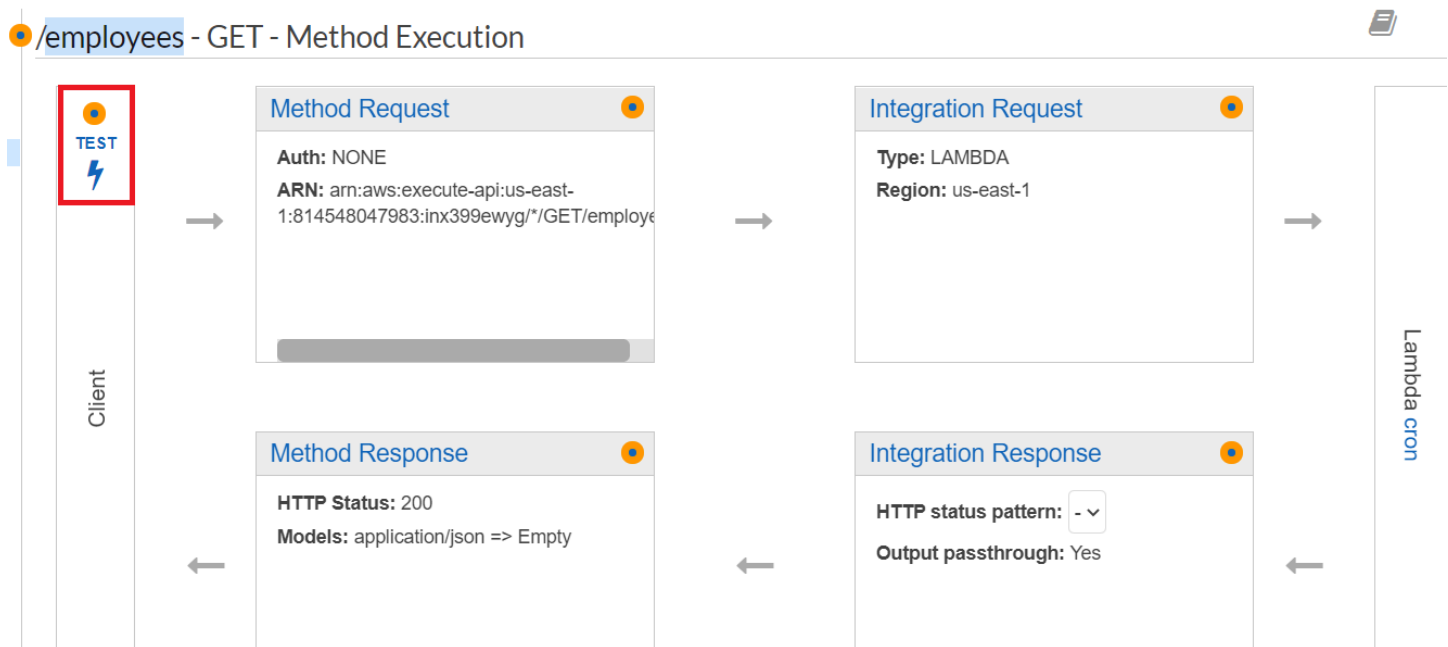
10. Wählen Sie `/employees`, wählen Sie in den Aktionen die Option Methode erstellen aus und wählen Sie anschließend im Drop-down-Menü unter `/employees` die Option GET aus. Wählen Sie das Häkchen aus.



11. Wählen Sie Lambda-Funktion und geben Sie mylambdafunction als Namen der Lambda-Funktion ein. Wählen Sie Speichern.

Testen Sie die API Gateway Gateway-Methode

An dieser Stelle des Tutorials können Sie die API Gateway Gateway-Methode testen, die die Lambda-Funktion mylambdafunction aufruft. Um die Methode zu testen, wählen Sie Test aus, wie in der folgenden Abbildung gezeigt.

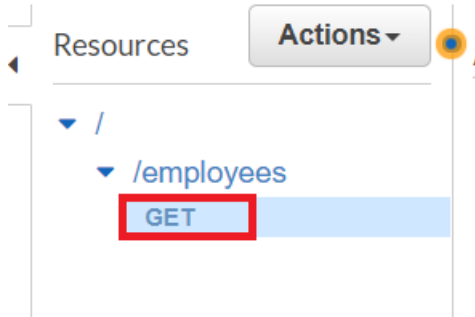


Sobald die Lambda-Funktion aufgerufen wurde, können Sie die Protokolldatei einsehen, um eine Erfolgsmeldung zu sehen.

Stellen Sie die API Gateway Gateway-Methode bereit

Nach erfolgreichem Test können Sie die Methode über die [Amazon API Gateway Gateway-Konsole](#) bereitstellen.

1. Wählen Sie Get.



2. Wählen Sie in der Dropdownliste Aktionen die Option API bereitstellen aus.

Deploy API

Choose a stage where your API will be deployed. For example, a test version of your API could be deployed to a stage named beta.

Deployment stage

[New Stage]

Stage name*

lambdastage

Stage description

Deployment description

Cancel

Deploy

3. Füllen Sie das Formular Deploy API aus und wählen Sie Deploy aus.

Deploy API

Choose a stage where your API will be deployed. For example, a test version of your API could be deployed to a stage named beta.

Deployment stage	[New Stage]
Stage name*	lambdastage
Stage description	
Deployment description	

Cancel Deploy

- Wählen Sie Save Changes.
- Wählen Sie erneut Abrufen und stellen Sie fest, dass sich die URL ändert. Dies ist die Aufruf-URL, mit der Sie die Lambda-Funktion aufrufen können.

Stages

Create

lambdastage

/

/employees

GET

lambdastage - GET - /employees

Invoke URL: <https://ewyg.execute-api.us-east-1.amazonaws.com/lambdastage/employees>

Use this page to override the [lambdastage stage](#) settings for the GET to /employees method.

Settings ☒ Inherit from stage ☐ Override for this method

Löschen Sie die Ressourcen

Herzlichen Glückwunsch! Sie haben eine Lambda-Funktion über Amazon API Gateway mit dem aufgerufen. AWS SDK für JavaScript Wie bereits zu Beginn dieses Tutorials erwähnt, sollten Sie darauf achten, alle Ressourcen zu beenden, die Sie während der Bearbeitung dieses Tutorials erstellen, um sicherzustellen, dass Ihnen nichts in Rechnung gestellt wird. Sie können dies tun, indem Sie den CloudFormation Stack, den Sie im [Erstellen Sie die AWS Ressourcen](#) Thema dieses Tutorials erstellt haben, wie folgt löschen:

- Öffnen Sie das [CloudFormation in der AWS Managementkonsole](#).

2. Öffnen Sie die Seite Stacks und wählen Sie den Stack aus.
3. Wählen Sie Löschen aus.

Geplante Ereignisse zur Ausführung von AWS Lambda Funktionen erstellen

Sie können ein geplantes Ereignis erstellen, das eine AWS Lambda Funktion aufruft, indem Sie ein CloudWatch Amazon-Ereignis verwenden. Sie können ein CloudWatch Ereignis so konfigurieren, dass ein Cron-Ausdruck verwendet wird, um zu planen, wann eine Lambda-Funktion aufgerufen wird. Sie können beispielsweise ein CloudWatch Ereignis planen, um an jedem Wochentag eine Lambda-Funktion aufzurufen.

AWS Lambda ist ein Rechendienst, mit dem Sie Code ausführen können, ohne Server bereitstellen oder verwalten zu müssen. Sie können Lambda-Funktionen in verschiedenen Programmiersprachen erstellen. Weitere Informationen zu finden Sie AWS Lambda unter [Was ist AWS Lambda](#).

In diesem Tutorial erstellen Sie eine Lambda-Funktion mithilfe der JavaScript Lambda-Laufzeit-API. In diesem Beispiel werden verschiedene AWS Dienste aufgerufen, um einen bestimmten Anwendungsfall auszuführen. Nehmen wir beispielsweise an, dass eine Organisation ihren Mitarbeitern eine mobile Textnachricht sendet, in der sie zum einjährigen Jubiläum gratuliert, wie in dieser Abbildung gezeigt.



Für dieses Tutorial benötigen Sie ungefähr 20 Minuten.

In diesem Tutorial erfahren Sie, wie Sie mithilfe von JavaScript Logik eine Lösung erstellen, die diesen Anwendungsfall erfüllt. Sie lernen beispielsweise, eine Datenbank zu lesen, um festzustellen, welche Mitarbeiter das einjährige Jubiläum erreicht haben, wie die Daten verarbeitet und eine Textnachricht mithilfe einer Lambda-Funktion versendet werden. Dann lernen Sie, wie Sie einen Cron-Ausdruck verwenden, um die Lambda-Funktion an jedem Wochentag aufzurufen.

AWS In diesem Tutorial wird eine Amazon DynamoDB-Tabelle mit dem Namen Employee verwendet, die diese Felder enthält.

- `id` — der Primärschlüssel für die Tabelle.
- `FirstName` — Vorname des Mitarbeiters.
- `Telefon` — Telefonnummer des Mitarbeiters.
- `StartDate` — Startdatum des Mitarbeiters.

Scan ▾

[Table] Employee: Id ▾

⬆

+ Add filter

Start search

☐	Id ⓘ ▲	first ▼	phone ▼	startDate ▼	
☐	1	Scott	15555555654	2019-12-20	
☐	2	Malcolm	15555555654	2019-12-17	
☐	55	Lam	15555555654	2019-12-19	

Important

Bearbeitungskosten: Die in diesem Dokument enthaltenen AWS Dienste sind im AWS kostenlosen Kontingent enthalten. Stellen Sie jedoch sicher, dass Sie alle Ressourcen beenden, nachdem Sie dieses Tutorial abgeschlossen haben, um sicherzustellen, dass Ihnen nichts in Rechnung gestellt wird.

Um die App zu erstellen:

1. [Vollständige Voraussetzungen](#)
2. [Erstellen Sie die AWS Ressourcen](#)
3. [Bereiten Sie das Browser-Skript vor](#)
4. [Lambda-Funktion erstellen und hochladen](#)
5. [Stellen Sie die Lambda-Funktion bereit](#)
6. [Führen Sie die App aus](#)
7. [Löschen Sie die Ressourcen](#)

Erforderliche Aufgaben

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Richten Sie die Projektumgebung ein, um diese TypeScript Beispiele für Node.js auszuführen, und installieren Sie die erforderlichen Module AWS SDK für JavaScript und Module von Drittanbietern. Folgen Sie den Anweisungen auf [GitHub](#).
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zur Bereitstellung einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter [Dateien mit gemeinsam genutzten Konfigurationen und Anmeldeinformationen](#) im AWS SDKs Referenzhandbuch zu Tools.

Erstellen Sie die AWS Ressourcen

Für dieses Tutorial sind die folgenden Ressourcen erforderlich.

- Eine Amazon DynamoDB-Tabelle mit dem Namen Employee mit einem Schlüssel namens Id und den in der vorherigen Abbildung gezeigten Feldern. Stellen Sie sicher, dass Sie die richtigen Daten eingeben, einschließlich eines gültigen Mobiltelefons, mit dem Sie diesen Anwendungsfall testen möchten. Weitere Informationen finden Sie unter [Tabelle erstellen](#).
- Eine IAM-Rolle mit angehängten Berechtigungen zur Ausführung von Lambda-Funktionen.
- Ein Amazon S3 S3-Bucket zum Hosten der Lambda-Funktion.

Sie können diese Ressourcen manuell erstellen, wir empfehlen jedoch, diese Ressourcen CloudFormation wie in diesem Tutorial beschrieben bereitzustellen.

Erstellen Sie die AWS Ressourcen mit CloudFormation

CloudFormation ermöglicht es Ihnen, AWS Infrastrukturbereitstellungen vorhersehbar und wiederholt zu erstellen und bereitzustellen. [Weitere Informationen zu CloudFormation finden Sie im AWS CloudFormation Benutzerhandbuch](#).

Um den CloudFormation Stack zu erstellen, verwenden Sie AWS CLI:

1. Installieren und konfigurieren Sie die AWS CLI folgenden Anweisungen im [AWS CLI Benutzerhandbuch](#).
2. Erstellen Sie eine Datei mit dem Namen `setup.yaml` im Stammverzeichnis Ihres Projektordners und kopieren Sie den Inhalt [hier GitHub](#) hinein.

 Note

Die CloudFormation Vorlage wurde unter Verwendung der [hier AWS CDK verfügbaren Datei](#) generiert GitHub. Weitere Informationen zu finden Sie im [AWS Cloud Development Kit \(AWS CDK\) Entwicklerhandbuch](#). AWS CDK

3. Führen Sie den folgenden Befehl von der Befehlszeile aus und **STACK_NAME** ersetzen Sie ihn durch einen eindeutigen Namen für den Stack.

 Important

Der Stack-Name muss innerhalb einer AWS Region und eines AWS Kontos eindeutig sein. Sie können bis zu 128 Zeichen angeben. Zahlen und Bindestriche sind zulässig.

```
aws cloudformation create-stack --stack-name STACK_NAME --template-body file://  
setup.yaml --capabilities CAPABILITY_IAM
```

Weitere Informationen zu den create-stack Befehlsparametern finden Sie in der [AWS CLI Befehlsreferenz](#) und im [CloudFormation Benutzerhandbuch](#).

Rufen Sie eine Liste der Ressourcen in der Konsole auf, indem Sie den Stack im CloudFormation Dashboard öffnen und die Registerkarte Ressourcen auswählen. Sie benötigen diese für das Tutorial.

4. Verwenden Sie bei der Erstellung des Stacks die, AWS SDK für JavaScript um die DynamoDB-Tabelle aufzufüllen, wie unter beschrieben. [Füllen Sie die DynamoDB-Tabelle aus](#)

Füllen Sie die DynamoDB-Tabelle aus

Um die Tabelle aufzufüllen, erstellen Sie zunächst ein Verzeichnis mit dem Namen `libs` und darin eine Datei mit dem Namen `dynamoClient.js`, und fügen Sie den folgenden Inhalt ein.

```
const { DynamoDBClient } = require( "@aws-sdk/client-dynamodb" );  
// Set the AWS Region.  
const REGION = "REGION"; // e.g. "us-east-1"  
// Create an Amazon DynamoDB service client object.  
const dynamoClient = new DynamoDBClient({region:REGION});
```

```
module.exports = { dynamoClient };
```

Dieser Code ist [hier verfügbar GitHub](#).

Erstellen Sie als Nächstes eine Datei mit dem Namen `populate-table.js` im Stammverzeichnis Ihres Projektordners und kopieren Sie den Inhalt [hier GitHub](#) hinein. Ersetzen Sie für eines der Elemente den Wert für die `phone` Eigenschaft durch eine gültige Handynummer im E.164-Format und den Wert für `startDate` durch das heutige Datum.

Führen Sie den folgenden Befehl von der Befehlszeile aus.

```
node populate-table.js
```

```
const {
  BatchWriteItemCommand } = require( "aws-sdk/client-dynamodb" );
const {dynamoClient} = require(  "../libs/dynamoClient" );
// Set the parameters.
const params = {
  RequestItems: {
    Employees: [
      {
        PutRequest: {
          Item: {
            id: { N: "1" },
            firstName: { S: "Bob" },
            phone: { N: "155555555555654" },
            startDate: { S: "2019-12-20" },
          },
        },
      },
      {
        PutRequest: {
          Item: {
            id: { N: "2" },
            firstName: { S: "Xing" },
            phone: { N: "155555555555653" },
            startDate: { S: "2019-12-17" },
          },
        },
      },
    ],
  },
}
```

```

    PutRequest: {
      Item: {
        id: { N: "55" },
        firstName: { S: "Harriette" },
        phone: { N: "155555555555652" },
        startDate: { S: "2019-12-19" },
      },
    },
  ],
},
};

export const run = async () => {
  try {
    const data = await dbclient.send(new BatchWriteItemCommand(params));
    console.log("Success", data);
  } catch (err) {
    console.log("Error", err);
  }
};
run();

```

Dieser Code ist [hier auf](#) verfügbar GitHub.

Die AWS Lambda Funktion erstellen

Konfigurieren des SDKs

Importieren Sie zuerst die erforderlichen AWS SDK für JavaScript (v3) Module und Befehle: `DynamoDBClient` und `DynamoDB ScanCommand` und `SNSClient` und den Amazon `PublishCommand` SNS-Befehl. Ersetzen Sie durch die **REGION** Region. AWS Dann berechne das heutige Datum und weise es einem Parameter zu. Erstellen Sie dann die Parameter für `ScanCommand`. Replace **TABLE_NAME** mit dem Namen der Tabelle, die Sie im [Erstellen Sie die AWS Ressourcen](#) Abschnitt dieses Beispiels erstellt haben.

Der folgende Codeausschnitt veranschaulicht diesen Schritt. (Das vollständige Beispiel finden Sie unter [Bündelung der Lambda-Funktion](#).)

```

"use strict";
// Load the required clients and commands.
const { DynamoDBClient, ScanCommand } = require("@aws-sdk/client-dynamodb");

```



```
const { SNSClient, PublishCommand } = require("@aws-sdk/client-sns");

//Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"

// Get today's date.
const today = new Date();
const dd = String(today.getDate()).padStart(2, "0");
const mm = String(today.getMonth() + 1).padStart(2, "0"); //January is 0!
const yyyy = today.getFullYear();
const date = yyyy + "-" + mm + "-" + dd;

// Set the parameters for the ScanCommand method.
const params = {
  // Specify which items in the results are returned.
  FilterExpression: "startDate = :topic",
  // Define the expression attribute value, which are substitutes for the values you
  want to compare.
  ExpressionAttributeValues: {
    ":topic": { S: date },
  },
  // Set the projection expression, which the the attributes that you want.
  ProjectionExpression: "firstName, phone",
  TableName: "TABLE_NAME",
};
```

Scannen der DynamoDB-Tabelle

Erstellen Sie zunächst eine `async/await` FunktionsendText, die aufgerufen wird, um eine Textnachricht mithilfe von Amazon SNS `PublishCommand` zu veröffentlichen. Fügen Sie dann ein `try` Blockmuster hinzu, das die DynamoDB-Tabelle nach Mitarbeitern durchsucht, deren heutiges Arbeitsjubiläum ansteht. Anschließend wird die `sendText` Funktion aufgerufen, diesen Mitarbeitern eine Textnachricht zu senden. Wenn ein Fehler auftritt, wird der `catch` Block aufgerufen.

Der folgende Codeausschnitt veranschaulicht diesen Schritt. (Das vollständige Beispiel finden Sie unter [Bündelung der Lambda-Funktion](#).)

```
exports.handler = async (event, context, callback) => {
  // Helper function to send message using Amazon SNS.
  async function sendText(textParams) {
    try {
      const data = await snsclient.send(new PublishCommand(textParams));
      console.log("Message sent");
    }
  }
}
```

```
    } catch (err) {
      console.log("Error, message not sent ", err);
    }
  }
  try {
    // Scan the table to check identify employees with work anniversary today.
    const data = await dbclient.send(new ScanCommand(params));
    data.Items.forEach(function (element, index, array) {
      const textParams = {
        PhoneNumber: element.phone.N,
        Message:
          "Hi " +
          element.firstName.S +
          "; congratulations on your work anniversary!",
      };
      // Send message using Amazon SNS.
      sendText(textParams);
    });
  } catch (err) {
    console.log("Error, could not scan table ", err);
  }
};
```

Bündelung der Lambda-Funktion

In diesem Thema wird beschrieben, wie Sie die `mylambdafunction.js` und die erforderlichen AWS SDK für JavaScript Module für dieses Beispiel in einer gebündelten Datei namens `bündeln.index.js`

1. Falls Sie es noch nicht getan haben, folgen Sie den Anweisungen [Erforderliche Aufgaben](#) für dieses Beispiel, um Webpack zu installieren.

Note

Informationen zu Webpack finden Sie unter [Bündeln Sie Anwendungen mit Webpack](#)

2. Führen Sie in der Befehlszeile den folgenden Befehl aus, um das JavaScript für dieses Beispiel in einer Datei namens `<index.js>` zu bündeln:

```
webpack mylamdbafunction.js --mode development --target node --devtool false --
output-library-target umd -o index.js
```

 Important

Beachten Sie, dass die Ausgabe benannt ist `index.js`. Dies liegt daran, dass Lambda-Funktionen einen `index.js` Handler haben müssen, um zu funktionieren.

3. Komprimieren Sie die gebündelte Ausgabedatei `index.js`, in eine ZIP-Datei mit dem Namen `my-lambda-function.zip`
4. Laden Sie `mylambdafunction.zip` es in den Amazon S3 S3-Bucket hoch, den Sie im [Erstellen Sie die AWS Ressourcen](#) Thema dieses Tutorials erstellt haben.

Hier ist der vollständige Browser-Skriptcode für `mylambdafunction.js`.

```
"use strict";
// Load the required clients and commands.
const { DynamoDBClient, ScanCommand } = require("@aws-sdk/client-dynamodb");
const { SNSClient, PublishCommand } = require("@aws-sdk/client-sns");

//Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"

// Get today's date.
const today = new Date();
const dd = String(today.getDate()).padStart(2, "0");
const mm = String(today.getMonth() + 1).padStart(2, "0"); //January is 0!
const yyyy = today.getFullYear();
const date = yyyy + "-" + mm + "-" + dd;

// Set the parameters for the ScanCommand method.
const params = {
  // Specify which items in the results are returned.
  FilterExpression: "startDate = :topic",
  // Define the expression attribute value, which are substitutes for the values you
  want to compare.
  ExpressionAttributeValues: {
    ":topic": { S: date },
  },
  // Set the projection expression, which the the attributes that you want.
  ProjectionExpression: "firstName, phone",
  TableName: "TABLE_NAME",
};
```

```
// Create the client service objects.
const dbclient = new DynamoDBClient({ region: REGION });
const snsclient = new SNSClient({ region: REGION });

exports.handler = async (event, context, callback) => {
  // Helper function to send message using Amazon SNS.
  async function sendText(textParams) {
    try {
      const data = await snsclient.send(new PublishCommand(textParams));
      console.log("Message sent");
    } catch (err) {
      console.log("Error, message not sent ", err);
    }
  }
  try {
    // Scan the table to check identify employees with work anniversary today.
    const data = await dbclient.send(new ScanCommand(params));
    data.Items.forEach(function (element, index, array) {
      const textParams = {
        PhoneNumber: element.phone.N,
        Message:
          "Hi " +
          element.firstName.S +
          "; congratulations on your work anniversary!",
      };
      // Send message using Amazon SNS.
      sendText(textParams);
    });
  } catch (err) {
    console.log("Error, could not scan table ", err);
  }
};
```

Stellen Sie die Lambda-Funktion bereit

Erstellen Sie im Stammverzeichnis Ihres Projekts eine `lambda-function-setup.js` Datei und fügen Sie den folgenden Inhalt ein.

`BUCKET_NAME` Ersetzen Sie es durch den Namen des Amazon S3 S3-Buckets, in den Sie die ZIP-Version Ihrer Lambda-Funktion hochgeladen haben. **`ZIP_FILE_NAME`** Ersetzen Sie die ZIP-Version Ihrer Lambda-Funktion durch den Namen oder Namen. **`IAM_ROLE_ARN`** Ersetzen Sie es durch die Amazon-Ressourcennummer (ARN) der IAM-Rolle, die Sie im [Erstellen Sie die AWS Ressourcen](#)

Thema dieses Tutorials erstellt haben. `LAMBDA_FUNCTION_NAME` Ersetzen Sie durch einen Namen für die Lambda-Funktion.

```
// Load the required Lambda client and commands.
const {
  CreateFunctionCommand,
} = require("@aws-sdk/client-lambda");
const {
  lambdaClient
} = require("../libs/lambdaClient.js");

// Instantiate an Lambda client service object.
const lambda = new LambdaClient({ region: REGION });

// Set the parameters.
const params = {
  Code: {
    S3Bucket: "BUCKET_NAME", // BUCKET_NAME
    S3Key: "ZIP_FILE_NAME", // ZIP_FILE_NAME
  },
  FunctionName: "LAMBDA_FUNCTION_NAME",
  Handler: "index.handler",
  Role: "IAM_ROLE_ARN", // IAM_ROLE_ARN; e.g., arn:aws:iam::650138640062:role/v3-
  lambda-tutorial-lambda-role
  Runtime: "nodejs12.x",
  Description:
    "Scans a DynamoDB table of employee details and using Amazon Simple Notification
    Services (Amazon SNS) to " +
    "send employees an email the each anniversary of their start-date.",
};

const run = async () => {
  try {
    const data = await lambda.send(new CreateFunctionCommand(params));
    console.log("Success", data); // successful response
  } catch (err) {
    console.log("Error", err); // an error occurred
  }
};
run();
```

Geben Sie in der Befehlszeile Folgendes ein, um die Lambda-Funktion bereitzustellen.

```
node lambda-function-setup.js
```

Dieses Codebeispiel ist [hier unter verfügbar GitHub](#).

So konfigurieren CloudWatch , dass die Lambda-Funktionen aufgerufen werden

So konfigurieren Sie CloudWatch das Aufrufen der Lambda-Funktionen:

1. Öffnen Sie die Seite Funktionen der Lambda-Konsole.
2. Wählen Sie die Lambda-Funktion.
3. Wählen Sie unter Designer die Option Add trigger (Trigger hinzufügen).
4. Stellen Sie den Triggertyp auf Events/ einCloudWatch . EventBridge
5. Wählen Sie für Regel die Option Neue Regel erstellen aus.
6. Geben Sie den Regelnamen und die Regelbeschreibung ein.
7. Wählen Sie als Regeltyp die Option Schedule expression aus.
8. Geben Sie im Feld Zeitplanausdruck einen Cron-Ausdruck ein. Zum Beispiel cron (0) 12? * MO-FR (*).
9. Wählen Sie Hinzufügen aus.

 Note

Weitere Informationen finden Sie unter [Lambda mit CloudWatch Ereignissen verwenden](#).

Löschen Sie die Ressourcen

Herzlichen Glückwunsch! Sie haben eine Lambda-Funktion über von Amazon CloudWatch geplante Ereignisse mit dem aufgerufen. AWS SDK für JavaScript Wie bereits zu Beginn dieses Tutorials erwähnt, sollten Sie darauf achten, alle Ressourcen zu beenden, die Sie während der Bearbeitung dieses Tutorials erstellen, um sicherzustellen, dass Ihnen nichts berechnet wird. Sie können dies tun, indem Sie den CloudFormation Stack, den Sie im [Erstellen Sie die AWS Ressourcen](#) Thema dieses Tutorials erstellt haben, wie folgt löschen:

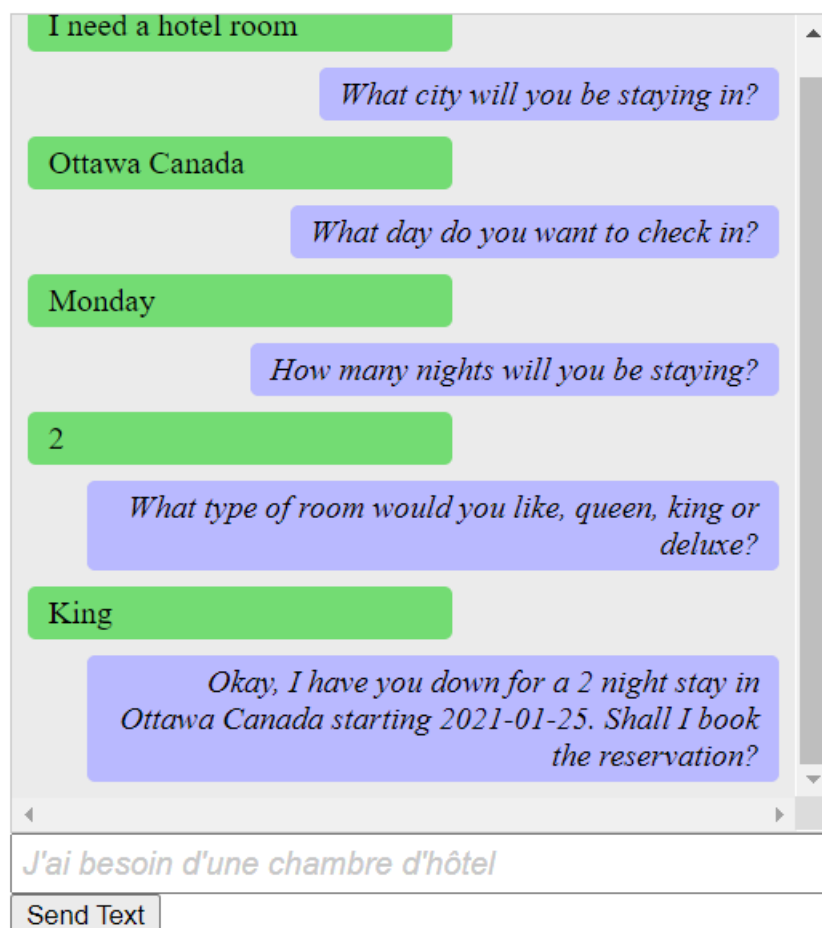
1. Öffnen Sie die [CloudFormation -Konsole](#).
2. Wählen Sie auf der Seite Stacks den Stapel aus.
3. Wählen Sie Löschen aus.

Erstellen eines Amazon-Lex-Chatbots

Sie können einen Amazon Lex-Chatbot in einer Webanwendung erstellen, um die Besucher Ihrer Website anzusprechen. Ein Amazon Lex Chatbot ist eine Funktion, die Online-Chat-Konversationen mit Benutzern durchführt, ohne direkten Kontakt zu einer Person herzustellen. Die folgende Abbildung zeigt beispielsweise einen Amazon Lex-Chatbot, der einen Benutzer mit der Buchung eines Hotelzimmers beauftragt.

Amazon Lex - BookTrip

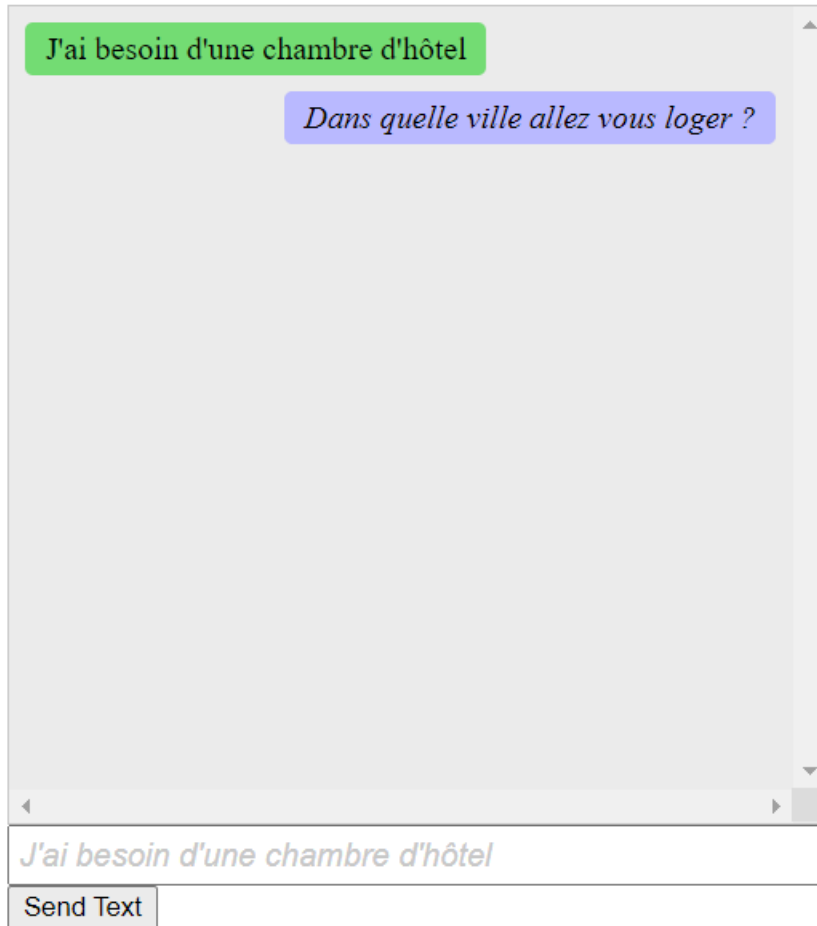
This multiple language chatbot shows you how easy it is to incorporate [Amazon Lex](#) into your web apps. Try it out.



Der in diesem AWS Tutorial erstellte Amazon Lex Chatbot kann mehrere Sprachen verarbeiten. Beispielsweise kann ein Benutzer, der Französisch spricht, französischen Text eingeben und eine Antwort auf Französisch erhalten.

Amazon Lex - BookTrip

This little chatbot shows how easy it is to incorporate [Amazon Lex](#) into your web pages. Try it out.



Ebenso kann ein Benutzer mit dem Amazon Lex Chatbot auf Italienisch kommunizieren.

Amazon Lex - BookTrip

This little chatbot shows how easy it is to incorporate [Amazon Lex](#) into your web pages. Try it out.



Dieses AWS Tutorial führt Sie durch die Erstellung eines Amazon Lex Lex-Chatbots und dessen Integration in eine Node.js Webanwendung. Die AWS SDK für JavaScript (v3) wird verwendet, um diese AWS Dienste aufzurufen:

- Amazon Lex
- Amazon Comprehend
- Amazon Translate

Bearbeitungskosten: Die in diesem Dokument enthaltenen AWS Dienste sind im [AWS kostenlosen Kontingent](#) enthalten.

Hinweis: Achten Sie darauf, alle Ressourcen, die Sie während der Bearbeitung dieses Tutorials erstellt haben, zu beenden, um sicherzustellen, dass Ihnen nichts in Rechnung gestellt wird.

Um die App zu erstellen:

1. [Voraussetzungen](#)
2. [Bereitstellen von Ressourcen](#)

3. [Amazon Lex Chatbot erstellen](#)
4. [Erstellen Sie den HTML-Code](#)
5. [Erstellen Sie das Browser-Skript](#)
6. [Nächste Schritte](#)

Voraussetzungen

Zum Einrichten und Ausführen dieses Beispiels müssen Sie zunächst diese Aufgaben abschließen:

- Richten Sie die Projektumgebung ein, um diese TypeScript Node-Beispiele auszuführen, und installieren Sie die erforderlichen Module AWS SDK für JavaScript und Module von Drittanbietern. Folgen Sie den Anweisungen auf [GitHub](#).
- Erstellen Sie eine freigegebene Konfigurationsdatei mit Ihren Anmeldeinformationen. Weitere Informationen zur Bereitstellung einer Datei mit gemeinsam genutzten Anmeldeinformationen finden Sie unter [Dateien mit gemeinsam genutzten Konfigurationen und Anmeldeinformationen](#) im AWS SDKs Referenzhandbuch zu Tools.

Important

In diesem Beispiel wird ECMAScript6 (ES6) verwendet. Dies erfordert Node.js Version 13.x oder höher. Informationen zum Herunterladen und Installieren der neuesten Version von Node.js finden Sie unter [Node.js downloads](#).

Wenn Sie jedoch die CommonJS-Syntax bevorzugen, finden Sie weitere Informationen unter [JavaScript ES6/CommonJs-Syntax](#).

Erstellen Sie die Ressourcen AWS

Für dieses Tutorial sind die folgenden Ressourcen erforderlich.

- Eine nicht authentifizierte IAM-Rolle mit angehängten Berechtigungen für:
 - Amazon Comprehend
 - Amazon Translate
 - Amazon Lex

Sie können diese Ressourcen manuell erstellen, wir empfehlen jedoch, diese Ressourcen AWS CloudFormation wie in diesem Tutorial beschrieben bereitzustellen.

Erstellen Sie die AWS Ressourcen mit CloudFormation

CloudFormation ermöglicht es Ihnen, AWS Infrastrukturbereitstellungen vorhersehbar und wiederholt zu erstellen und bereitzustellen. [Weitere Informationen zu CloudFormation finden Sie im AWS CloudFormation Benutzerhandbuch.](#)

Um den CloudFormation Stack zu erstellen, verwenden Sie AWS CLI:

1. Installieren und konfigurieren Sie die AWS CLI folgenden Anweisungen im [AWS CLI Benutzerhandbuch](#).
2. Erstellen Sie eine Datei mit dem Namen `setup.yaml` im Stammverzeichnis Ihres Projektordners und kopieren Sie den Inhalt [hier GitHub](#) hinein.

 Note

Die CloudFormation Vorlage wurde unter Verwendung der [hier AWS CDK verfügbaren Datei](#) generiert GitHub. Weitere Informationen zu finden Sie im [AWS Cloud Development Kit \(AWS CDK\) Entwicklerhandbuch](#). AWS CDK

3. Führen Sie den folgenden Befehl von der Befehlszeile aus und `STACK_NAME` ersetzen Sie ihn durch einen eindeutigen Namen für den Stack.

 Important

Der Stack-Name muss innerhalb einer AWS Region und eines AWS Kontos eindeutig sein. Sie können bis zu 128 Zeichen angeben. Zahlen und Bindestriche sind zulässig.

```
aws cloudformation create-stack --stack-name STACK_NAME --template-body file://  
setup.yaml --capabilities CAPABILITY_IAM
```

Weitere Informationen zu den `create-stack` Befehlsparametern finden Sie in der [AWS CLI Befehlsreferenz](#) und im [CloudFormation Benutzerhandbuch](#).

Um die erstellten Ressourcen anzuzeigen, öffnen Sie die Amazon Lex Lex-Konsole, wählen Sie den Stack und dann die Registerkarte Ressourcen aus.

Erstellen Sie einen Amazon-Lex-Bot

⚠ Important

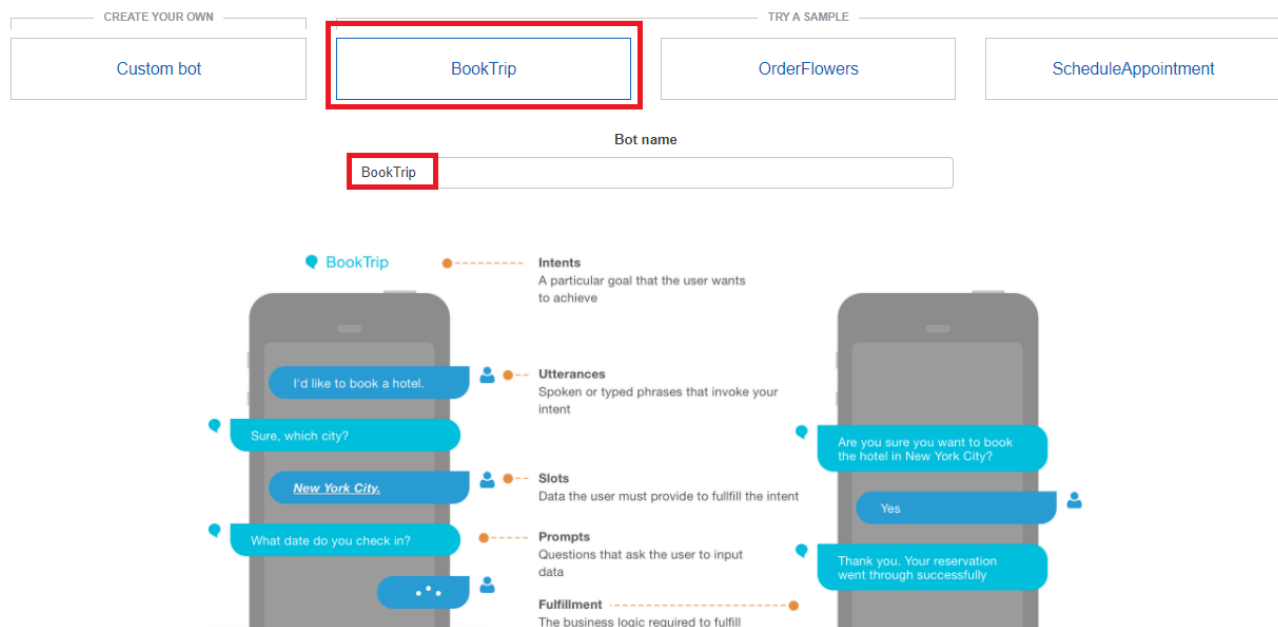
Verwenden Sie V1 der Amazon Lex-Konsole, um den Bot zu erstellen. Dieses Beispiel funktioniert nicht mit Bots, die mit V2 erstellt wurden.

Der erste Schritt besteht darin, mithilfe der Amazon Web Services Management Console einen Amazon Lex-Chatbot zu erstellen. In diesem Beispiel wird das Amazon Lex BookTripLex-Beispiel verwendet. Weitere Informationen finden Sie unter [Reise buchen](#).

- Melden Sie sich bei der Amazon Web Services Management Console an und öffnen Sie die Amazon Lex Lex-Konsole in der [Amazon Web Services Console](#).
- Wählen Sie auf der Seite Bots die Option Erstellen aus.
- Wählen Sie BookTripBlueprint (behalten Sie den Standard-Bot-Namen bei BookTrip).

Create your bot

Amazon Lex enables any developer to build conversational chatbots quickly and easily. With Amazon Lex, no deep learning expertise is necessary—you just specify the basic conversational flow directly from the console, and then Amazon Lex manages the dialogue and dynamically adjusts the response. To get started, you can choose one of the sample bots provided below or build a new custom bot from scratch.



- Füllen Sie die Standardeinstellungen aus und wählen Sie Erstellen (die Konsole zeigt den BookTripBot). Überprüfen Sie auf der Registerkarte Editor die Details der vorkonfigurierten Absichten.

- Testen Sie den Bot im Testfenster. Starten Sie den Test, indem Sie Ich möchte ein Hotelzimmer buchen eingeben.

> Test bot (Latest)

✓ Ready. Build complete

I want to book a hotel room

What city will you be staying in?

Clear chat history



Chat with your bot...

Inspect response

Dialog State: ElicitSlot

Hide



Summary



Detail

Intent: BookHotel

- Wählen Sie Veröffentlichen und geben Sie einen Aliasnamen an (Sie benötigen diesen Wert, wenn Sie den verwenden AWS SDK für JavaScript).

Note

Sie müssen in Ihrem JavaScript Code auf den Bot-Namen und den Bot-Alias verweisen.

Erstellen Sie den HTML-Code

Erstellen Sie eine Datei namens `index.html`. Kopieren Sie den folgenden Code und fügen Sie ihn in `index.html`. Dieser HTML-Verweis `main.js`. Dies ist eine gebündelte Version von `index.js`, die die erforderlichen AWS SDK für JavaScript Module enthält. Sie werden diese Datei in [Erstellen Sie den HTML-Code](#) erstellen. `index.html` auch `Referenzenstyle.css`, wodurch die Stile hinzugefügt werden.

```
<!doctype html>
<head>
  <title>Amazon Lex - Sample Application (BookTrip)</title>
  <link type="text/css" rel="stylesheet" href="style.css" />
</head>

<body>
  <h1 id="title">Amazon Lex - BookTrip</h1>
  <p id="intro">
    This multiple language chatbot shows you how easy it is to incorporate
    <a
      href="https://aws.amazon.com/lex/"
      title="Amazon Lex (product)"
      target="_new"
    >Amazon Lex</a>
    >
    into your web apps. Try it out.
  </p>
  <div id="conversation"></div>
  <input
    type="text"
    id="wisdom"
    size="80"
    value=""
    placeholder="J'ai besoin d'une chambre d'hôtel"
  />
  <br />
  <button onclick="createResponse()">Send Text</button>
  <script type="text/javascript" src="./main.js"></script>
</body>
```

Dieser Code ist auch [hier verfügbar GitHub](#).

Erstellen Sie das Browser-Skript

Erstellen Sie eine Datei namens `index.js`. Kopieren Sie den folgenden Code und fügen Sie ihn ein `index.js`. Importieren Sie die erforderlichen AWS SDK für JavaScript Module und Befehle. Erstellen Sie Clients für Amazon Lex, Amazon Comprehend und Amazon Translate. **REGION** Ersetzen Sie durch AWS Region und **IDENTITY_POOL_ID** durch die ID des Identitätspools, den Sie in der erstellt haben. [Erstellen Sie die Ressourcen AWS](#) Um diese Identitätspool-ID abzurufen, öffnen Sie den Identitätspool in der Amazon Cognito Cognito-Konsole, wählen Sie Identitätspool bearbeiten

und dann im Seitenmenü Beispielcode aus. Die Identitätspool-ID wird in der Konsole in rotem Text angezeigt.

Erstellen Sie zunächst ein `libs` Verzeichnis und erstellen Sie die erforderlichen Service-Client-Objekte, indem Sie drei Dateien, `comprehendClient.js`, `lexClient.js`, und `erstellentranslateClient.js`. Fügen Sie den entsprechenden Code unten in jede Datei ein `REGION` und ersetzen Sie sie `IDENTITY_POOL_ID` in jeder Datei.

Note

Verwenden Sie die ID des Amazon Cognito Cognito-Identitätspools, in [Erstellen Sie die AWS Ressourcen mit CloudFormation](#) dem Sie ihn erstellt haben.

```
import { CognitoIdentityClient } from "@aws-sdk/client-cognito-identity";
import { fromCognitoIdentityPool } from "@aws-sdk/credential-provider-cognito-identity";
import { ComprehendClient } from "@aws-sdk/client-comprehend";

const REGION = "REGION";
const IDENTITY_POOL_ID = "IDENTITY_POOL_ID"; // An Amazon Cognito Identity Pool ID.

// Create an Amazon Comprehend service client object.
const comprehendClient = new ComprehendClient({
  region: REGION,
  credentials: fromCognitoIdentityPool({
    client: new CognitoIdentityClient({ region: REGION }),
    identityPoolId: IDENTITY_POOL_ID,
  }),
});

export { comprehendClient };
```

```
import { CognitoIdentityClient } from "@aws-sdk/client-cognito-identity";
import { fromCognitoIdentityPool } from "@aws-sdk/credential-provider-cognito-identity";
import { LexRuntimeServiceClient } from "@aws-sdk/client-lex-runtime-service";

const REGION = "REGION";
const IDENTITY_POOL_ID = "IDENTITY_POOL_ID"; // An Amazon Cognito Identity Pool ID.
```

```
// Create an Amazon Lex service client object.
const lexClient = new LexRuntimeServiceClient({
  region: REGION,
  credentials: fromCognitoIdentityPool({
    client: new CognitoIdentityClient({ region: REGION }),
    identityPoolId: IDENTITY_POOL_ID,
  }),
});

export { lexClient };
```

```
import { CognitoIdentityClient } from "@aws-sdk/client-cognito-identity";
import { fromCognitoIdentityPool } from "@aws-sdk/credential-provider-cognito-identity";
import { TranslateClient } from "@aws-sdk/client-translate";

const REGION = "REGION";
const IDENTITY_POOL_ID = "IDENTITY_POOL_ID"; // An Amazon Cognito Identity Pool ID.

// Create an Amazon Translate service client object.
const translateClient = new TranslateClient({
  region: REGION,
  credentials: fromCognitoIdentityPool({
    client: new CognitoIdentityClient({ region: REGION }),
    identityPoolId: IDENTITY_POOL_ID,
  }),
});

export { translateClient };
```

Dieser Code ist [hier auf GitHub verfügbar](#).

Erstellen Sie als Nächstes eine `index.js` Datei und fügen Sie den folgenden Code ein.

Ersetzen Sie ***BOT_ALIAS*** und ***BOT_NAME*** durch den Alias bzw. den Namen Ihres Amazon Lex Lex-Bot und ***USER_ID*** durch eine Benutzer-ID. Die `createResponse` asynchrone Funktion macht Folgendes:

- Nimmt den vom Benutzer in den Browser eingegebenen Text auf und verwendet Amazon Comprehend, um seinen Sprachcode zu ermitteln.
- Nimmt den Sprachcode und verwendet Amazon Translate, um den Text ins Englische zu übersetzen.

- Nimmt den übersetzten Text und verwendet Amazon Lex, um eine Antwort zu generieren.
- Sendet die Antwort auf der Browserseite.

```
import { DetectDominantLanguageCommand } from "@aws-sdk/client-comprehend";
import { TranslateTextCommand } from "@aws-sdk/client-translate";
import { PostTextCommand } from "@aws-sdk/client-lex-runtime-service";
import { lexClient } from "../libs/lexClient.js";
import { translateClient } from "../libs/translateClient.js";
import { comprehendClient } from "../libs/comprehendClient.js";

let g_text = "";
// Set the focus to the input box.
document.getElementById("wisdom").focus();

function showRequest() {
  const conversationDiv = document.getElementById("conversation");
  const requestPara = document.createElement("P");
  requestPara.className = "userRequest";
  requestPara.appendChild(document.createTextNode(g_text));
  conversationDiv.appendChild(requestPara);
  conversationDiv.scrollTop = conversationDiv.scrollHeight;
}

function showResponse(lexResponse) {
  const conversationDiv = document.getElementById("conversation");
  const responsePara = document.createElement("P");
  responsePara.className = "lexResponse";

  const lexTextResponse = lexResponse;

  responsePara.appendChild(document.createTextNode(lexTextResponse));
  responsePara.appendChild(document.createElement("br"));
  conversationDiv.appendChild(responsePara);
  conversationDiv.scrollTop = conversationDiv.scrollHeight;
}

function handletext(text) {
  g_text = text;
  const xhr = new XMLHttpRequest();
  xhr.addEventListener("load", loadNewItems, false);
  xhr.open("POST", "../text", true); // A Spring MVC controller
```

```
xhr.setRequestHeader("Content-type", "application/x-www-form-urlencoded"); //
necessary
  xhr.send(`text=${text}`);
}

function loadNewItems() {
  showRequest();

  // Re-enable input.
  const wisdomText = document.getElementById("wisdom");
  wisdomText.value = "";
  wisdomText.locked = false;
}

// Respond to user's input.
const createResponse = async () => {
  // Confirm there is text to submit.
  const wisdomText = document.getElementById("wisdom");
  if (wisdomText?.value && wisdomText.value.trim().length > 0) {
    // Disable input to show it is being sent.
    const wisdom = wisdomText.value.trim();
    wisdomText.value = "...";
    wisdomText.locked = true;
    handleText(wisdom);

    const comprehendParams = {
      Text: wisdom,
    };
    try {
      const data = await comprehendClient.send(
        new DetectDominantLanguageCommand(comprehendParams),
      );
      console.log(
        "Success. The language code is: ",
        data.Languages[0].LanguageCode,
      );
      const translateParams = {
        SourceLanguageCode: data.Languages[0].LanguageCode,
        TargetLanguageCode: "en", // For example, "en" for English.
        Text: wisdom,
      };
      try {
        const data = await translateClient.send(
          new TranslateTextCommand(translateParams),
```

```
);
console.log("Success. Translated text: ", data.TranslatedText);
const lexParams = {
  botName: "BookTrip",
  botAlias: "mynewalias",
  inputText: data.TranslatedText,
  userId: "chatbot", // For example, 'chatbot-demo'.
};
try {
  const data = await lexClient.send(new PostTextCommand(lexParams));
  console.log("Success. Response is: ", data.message);
  const msg = data.message;
  showResponse(msg);
} catch (err) {
  console.log("Error responding to message. ", err);
}
} catch (err) {
  console.log("Error translating text. ", err);
}
} catch (err) {
  console.log("Error identifying language. ", err);
}
}
};
// Make the function available to the browser.
window.createResponse = createResponse;
```

Dieser Code ist [hier verfügbar GitHub](#).

Verwenden Sie jetzt Webpack, um die AWS SDK für JavaScript Module `index.js` und in einer einzigen Datei zu bündeln. `main.js`

1. Falls Sie dies noch nicht getan haben, folgen Sie den Anweisungen in diesem [Voraussetzungen](#) Beispiel, um das Webpack zu installieren.

 Note

Informationen zu Webpack finden Sie unter. [Bündeln Sie Anwendungen mit Webpack](#)

2. Führen Sie in der Befehlszeile den folgenden Befehl aus, um das JavaScript für dieses Beispiel in einer Datei namens `main.js` zu bündeln:

```
webpack index.js --mode development --target web --devtool false -o main.js
```

Nächste Schritte

Herzlichen Glückwunsch! Sie haben eine Anwendung Node.js erstellt, die Amazon Lex verwendet, um ein interaktives Benutzererlebnis zu schaffen. Wie bereits zu Beginn dieses Tutorials erwähnt, sollten Sie darauf achten, alle Ressourcen, die Sie während der Bearbeitung dieses Tutorials erstellen, zu beenden, um sicherzustellen, dass Ihnen nichts in Rechnung gestellt wird. Sie können dies tun, indem Sie den CloudFormation Stack, den Sie im [Erstellen Sie die Ressourcen AWS](#) Thema dieses Tutorials erstellt haben, wie folgt löschen:

1. Öffnen Sie die [CloudFormation -Konsole](#).
2. Wählen Sie auf der Seite Stacks den Stapel aus.
3. Wählen Sie Löschen aus.

Weitere AWS dienstübergreifende Beispiele finden Sie unter [AWS SDK für JavaScript serviceübergreifende](#) Beispiele.

SDK für JavaScript (v3) -Codebeispiele

Die Codebeispiele in diesem Thema zeigen Ihnen, wie Sie AWS SDK für JavaScript (v3) mit verwenden AWS.

Bei Grundlagen handelt es sich um Codebeispiele, die Ihnen zeigen, wie Sie die wesentlichen Vorgänge innerhalb eines Services ausführen.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien anzeigen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb eines Services aufrufen oder mit anderen AWS-Services kombinieren.

Einige Services enthalten zusätzliche Beispielkategorien, die zeigen, wie Bibliotheken oder dienstspezifische Funktionen genutzt werden können.

Dienstleistungen

- [API Gateway Gateway-Beispiele mit SDK für JavaScript \(v3\)](#)
- [Aurora-Beispiele mit SDK für JavaScript \(v3\)](#)
- [Auto Scaling Scaling-Beispiele mit SDK für JavaScript \(v3\)](#)
- [Amazon Bedrock-Beispiele mit SDK für JavaScript \(v3\)](#)
- [Amazon Bedrock Runtime-Beispiele mit SDK für JavaScript \(v3\)](#)
- [Beispiele für Amazon Bedrock Agents mit SDK für JavaScript \(v3\)](#)
- [Amazon Bedrock Agents Runtime-Beispiele mit SDK für JavaScript \(v3\)](#)
- [CloudWatch Beispiele für die Verwendung von SDK für JavaScript \(v3\)](#)
- [CloudWatch Beispiele für Ereignisse mit SDK für JavaScript \(v3\)](#)
- [CloudWatch Protokolliert Beispiele mit SDK für JavaScript \(v3\)](#)
- [CodeBuild Beispiele für die Verwendung von SDK für JavaScript \(v3\)](#)
- [Beispiele für Amazon Cognito Identity mit SDK für JavaScript \(v3\)](#)
- [Beispiele für Amazon Cognito Identity Provider mit SDK für JavaScript \(v3\)](#)
- [Amazon Comprehend Comprehend-Beispiele mit SDK für JavaScript \(v3\)](#)

- [Amazon DocumentDB DocumentDB-Beispiele mit SDK für JavaScript \(v3\)](#)
- [DynamoDB-Beispiele mit SDK für JavaScript \(v3\)](#)
- [EC2 Amazon-Beispiele mit SDK für JavaScript \(v3\)](#)
- [Elastic Load Balancing — Beispiele für Version 2 mit SDK für JavaScript \(v3\)](#)
- [AWS Entity Resolution Beispiele mit SDK für JavaScript \(v3\)](#)
- [EventBridge Beispiele für die Verwendung von SDK für JavaScript \(v3\)](#)
- [Amazon Glacier-Beispiele mit SDK für JavaScript \(v3\)](#)
- [AWS Glue Beispiele für die Verwendung von SDK für JavaScript \(v3\)](#)
- [HealthImaging Beispiele für die Verwendung von SDK für JavaScript \(v3\)](#)
- [IAM-Beispiele mit SDK für JavaScript \(v3\)](#)
- [AWS IoT SiteWise Beispiele mit SDK für JavaScript \(v3\)](#)
- [Kinesis-Beispiele mit SDK für JavaScript \(v3\)](#)
- [Lambda-Beispiele mit SDK für JavaScript \(v3\)](#)
- [Amazon Lex Lex-Beispiele mit SDK für JavaScript \(v3\)](#)
- [Beispiele für Amazon-Standorte mit SDK für JavaScript \(v3\)](#)
- [Amazon MSK-Beispiele mit SDK für JavaScript \(v3\)](#)
- [Amazon Personalize Personalize-Beispiele mit SDK für JavaScript \(v3\)](#)
- [Beispiele für Amazon Personalize Events mit SDK für JavaScript \(v3\)](#)
- [Amazon Personalize Runtime-Beispiele mit SDK für JavaScript \(v3\)](#)
- [Amazon Pinpoint Pinpoint-Beispiele mit SDK für JavaScript \(v3\)](#)
- [Amazon Polly Polly-Beispiele mit SDK für JavaScript \(v3\)](#)
- [Amazon RDS-Beispiele mit SDK für JavaScript \(v3\)](#)
- [Beispiele für Amazon RDS Data Service mit SDK für JavaScript \(v3\)](#)
- [Amazon Redshift Redshift-Beispiele mit SDK für JavaScript \(v3\)](#)
- [Amazon Rekognition Rekognition-Beispiele mit SDK für JavaScript \(v3\)](#)
- [Amazon S3 S3-Beispiele mit SDK für JavaScript \(v3\)](#)
- [SageMaker KI-Beispiele mit SDK für JavaScript \(v3\)](#)
- [Secrets Manager Manager-Beispiele mit SDK für JavaScript \(v3\)](#)
- [Amazon SES SES-Beispiele mit SDK für JavaScript \(v3\)](#)

- [Amazon SNS SNS-Beispiele mit SDK für JavaScript \(v3\)](#)
- [Amazon SQS SQS-Beispiele mit SDK für JavaScript \(v3\)](#)
- [Step Functions Functions-Beispiele mit SDK für JavaScript \(v3\)](#)
- [AWS STS Beispiele für die Verwendung von SDK für JavaScript \(v3\)](#)
- [Support Beispiele für die Verwendung von SDK für JavaScript \(v3\)](#)
- [Systems Manager Manager-Beispiele mit SDK für JavaScript \(v3\)](#)
- [Amazon Textract Textract-Beispiele mit SDK für JavaScript \(v3\)](#)
- [Amazon Transcribe Transcribe-Beispiele mit SDK für JavaScript \(v3\)](#)
- [Amazon Translate Translate-Beispiele mit SDK für JavaScript \(v3\)](#)

API Gateway Gateway-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK für JavaScript (v3) mit API Gateway Aktionen ausführen und allgemeine Szenarien implementieren.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb eines Service aufrufen oder mit anderen AWS-Services kombinieren.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kodex finden.

Themen

- [Szenarien](#)

Szenarien

Erstellen einer Serverless-Anwendung zur Verwaltung von Fotos

Das folgende Codebeispiel zeigt, wie eine Serverless-Anwendung erstellt wird, mit der Benutzer Fotos mithilfe von Labels erstellen können.

SDK für JavaScript (v3)

Zeigt, wie eine Anwendung zur Verwaltung von Fotobeständen entwickelt wird, die mithilfe von Amazon Rekognition Labels in Bildern erkennt und sie für einen späteren Abruf speichert.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

Einen tiefen Einblick in den Ursprung dieses Beispiels finden Sie im Beitrag in der [AWS - Community](#).

In diesem Beispiel verwendete Dienste

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Verwenden von API Gateway zum Aufrufen einer Lambda-Funktion

Das folgende Codebeispiel zeigt, wie eine von Amazon API Gateway aufgerufene AWS Lambda Funktion erstellt wird.

SDK für JavaScript (v3)

Zeigt, wie eine AWS Lambda Funktion mithilfe der JavaScript Lambda-Laufzeit-API erstellt wird. In diesem Beispiel werden verschiedene AWS Dienste aufgerufen, um einen bestimmten Anwendungsfall auszuführen. Dieses Beispiel zeigt, wie man eine Lambda-Funktion erstellt, die von Amazon API Gateway aufgerufen wird und eine Amazon-DynamoDB-Tabelle nach Arbeitsjubiläen durchsucht und Amazon Simple Notification Service (Amazon SNS) verwendet, um eine Textnachricht an Ihre Mitarbeiter zu senden, die ihnen zu ihrem einjährigen Jubiläum gratuliert.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

Dieses Beispiel ist auch verfügbar im [AWS SDK für JavaScript Entwicklerhandbuch für v3](#).

In diesem Beispiel verwendete Dienste

- API Gateway

- DynamoDB
- Lambda
- Amazon SNS

Aurora-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK für JavaScript (v3) mit Aurora Aktionen ausführen und allgemeine Szenarien implementieren.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb eines Service aufrufen oder mit anderen AWS-Services kombinieren.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kodex finden.

Themen

- [Szenarien](#)

Szenarien

Erstellen eines Trackers für Aurora-Serverless-Arbeitsaufgaben

Das folgende Codebeispiel zeigt, wie eine Webanwendung erstellt wird, die Arbeitselemente in einer Datenbank von Amazon Aurora Serverless verfolgt und mithilfe von Amazon Simple Email Service (Amazon SES) Berichte sendet.

SDK für JavaScript (v3)

Zeigt, wie Sie mit AWS SDK für JavaScript (v3) eine Webanwendung erstellen, die Arbeitselemente in einer Amazon Aurora Aurora-Datenbank verfolgt und Berichte mithilfe von Amazon Simple Email Service (Amazon SES) per E-Mail versendet. In diesem Beispiel wird ein mit React.js erstelltes Frontend verwendet, um mit einem Express-Node.js-Backend zu interagieren.

- Integrieren Sie eine React.js Webanwendung mit AWS-Services.
- Auflisten, hinzufügen und aktualisieren von Elementen in einer Aurora-Tabelle.
- Senden Sie einen E-Mail-Bericht über gefilterte Arbeitselemente mit Amazon SES.

- Stellen Sie Beispielressourcen mit dem mitgelieferten AWS CloudFormation Skript bereit und verwalten Sie sie.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

In diesem Beispiel verwendete Dienste

- Aurora
- Amazon RDS
- Amazon RDS Data Service
- Amazon SES

Auto Scaling Scaling-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK für JavaScript (v3) mit Auto Scaling Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien anzeigen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb eines Service aufrufen oder mit anderen AWS-Services kombinieren.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kodex finden.

Themen

- [Aktionen](#)
- [Szenarien](#)

Aktionen

AttachLoadBalancerTargetGroups

Das folgende Codebeispiel zeigt die Verwendung von `AttachLoadBalancerTargetGroups`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
const client = new AutoScalingClient({});
await client.send(
  new AttachLoadBalancerTargetGroupsCommand({
    AutoScalingGroupName: NAMES.autoScalingGroupName,
    TargetGroupARNs: [state.targetGroupArn],
  }),
);
```

- Einzelheiten zur API finden Sie [AttachLoadBalancerTargetGroups](#) in der AWS SDK für JavaScript API-Referenz.

Szenarien

Erstellen und Verwalten eines ausfallsicheren Services

Das folgende Codebeispiel zeigt, wie Sie einen Webservice mit Load Balancing erstellen, der Buch-, Film- und Liedempfehlungen zurückgibt. Das Beispiel zeigt, wie der Service auf Fehler reagiert und wie der Service für mehr Ausfallsicherheit umstrukturiert werden kann.

- Verwenden Sie eine Amazon EC2 Auto Scaling Scaling-Gruppe, um Amazon Elastic Compute Cloud (Amazon EC2) -Instances auf der Grundlage einer Startvorlage zu erstellen und die Anzahl der Instances in einem bestimmten Bereich zu halten.
- Verarbeiten und verteilen Sie HTTP-Anfragen mit Elastic Load Balancing.
- Überwachen Sie den Zustand von Instances in einer Auto-Scaling-Gruppe und leiten Sie Anfragen nur an fehlerfreie Instances weiter.
- Führen Sie auf jeder EC2 Instanz einen Python-Webserver aus, um HTTP-Anfragen zu verarbeiten. Der Webserver reagiert mit Empfehlungen und Zustandsprüfungen.
- Simulieren Sie einen Empfehlungsservice mit einer Amazon DynamoDB-Tabelle.

- Steuern Sie die Antwort des Webserver auf Anfragen und Zustandsprüfungen, indem Sie die AWS Systems Manager Parameter aktualisieren.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Führen Sie ein interaktives Szenario an einer Eingabeaufforderung aus.

```
#!/usr/bin/env node
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import {
  Scenario,
  parseScenarioArgs,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * The workflow steps are split into three stages:
 *   - deploy
 *   - demo
 *   - destroy
 *
 * Each of these stages has a corresponding file prefixed with steps-*.
 */
import { deploySteps } from "./steps-deploy.js";
import { demoSteps } from "./steps-demo.js";
import { destroySteps } from "./steps-destroy.js";

/**
 * The context is passed to every scenario. Scenario steps
 * will modify the context.
 */
const context = {};
```

```

* Three Scenarios are created for the workflow. A Scenario is an orchestration
class
* that simplifies running a series of steps.
*/
export const scenarios = {
  // Deploys all resources necessary for the workflow.
  deploy: new Scenario("Resilient Workflow - Deploy", deploySteps, context),
  // Demonstrates how a fragile web service can be made more resilient.
  demo: new Scenario("Resilient Workflow - Demo", demoSteps, context),
  // Destroys the resources created for the workflow.
  destroy: new Scenario("Resilient Workflow - Destroy", destroySteps, context),
};

// Call function if run directly
import { fileURLToPath } from "node:url";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios, {
    name: "Resilient Workflow",
    synopsis:
      "node index.js --scenario <deploy | demo | destroy> [-h|--help] [-y|--yes] [-v|--verbose]",
    description: "Deploy and interact with scalable EC2 instances.",
  });
}

```

Erstellen Sie Schritte, um alle Ressourcen bereitzustellen.

```

import { join } from "node:path";
import { readFileSync, writeFileSync } from "node:fs";
import axios from "axios";

import {
  BatchWriteItemCommand,
  CreateTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  CreateKeyPairCommand,
  CreateLaunchTemplateCommand,

```

```

    DescribeAvailabilityZonesCommand,
    DescribeVpcsCommand,
    DescribeSubnetsCommand,
    DescribeSecurityGroupsCommand,
    AuthorizeSecurityGroupIngressCommand,
} from "@aws-sdk/client-ec2";
import {
    IAMClient,
    CreatePolicyCommand,
    CreateRoleCommand,
    CreateInstanceProfileCommand,
    AddRoleToInstanceProfileCommand,
    AttachRolePolicyCommand,
    waitUntilInstanceProfileExists,
} from "@aws-sdk/client-iam";
import { SSMClient, GetParameterCommand } from "@aws-sdk/client-ssm";
import {
    CreateAutoScalingGroupCommand,
    AutoScalingClient,
    AttachLoadBalancerTargetGroupsCommand,
} from "@aws-sdk/client-auto-scaling";
import {
    CreateListenerCommand,
    CreateLoadBalancerCommand,
    CreateTargetGroupCommand,
    ElasticLoadBalancingV2Client,
    waitUntilLoadBalancerAvailable,
} from "@aws-sdk/client-elastic-load-balancing-v2";

import {
    ScenarioOutput,
    ScenarioInput,
    ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { saveState } from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES, RESOURCES_PATH, ROOT } from "../constants.js";
import { initParamsSteps } from "../steps-reset-params.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const deploySteps = [

```

```
new ScenarioOutput("introduction", MESSAGES.introduction, { header: true }),
new ScenarioInput("confirmDeployment", MESSAGES.confirmDeployment, {
  type: "confirm",
}),
new ScenarioAction(
  "handleConfirmDeployment",
  (c) => c.confirmDeployment === false && process.exit(),
),
new ScenarioOutput(
  "creatingTable",
  MESSAGES.creatingTable.replace("${TABLE_NAME}", NAMES.tableName),
),
new ScenarioAction("createTable", async () => {
  const client = new DynamoDBClient({});
  await client.send(
    new CreateTableCommand({
      TableName: NAMES.tableName,
      ProvisionedThroughput: {
        ReadCapacityUnits: 5,
        WriteCapacityUnits: 5,
      },
      AttributeDefinitions: [
        {
          AttributeName: "MediaType",
          AttributeType: "S",
        },
        {
          AttributeName: "ItemId",
          AttributeType: "N",
        },
      ],
      KeySchema: [
        {
          AttributeName: "MediaType",
          KeyType: "HASH",
        },
        {
          AttributeName: "ItemId",
          KeyType: "RANGE",
        },
      ],
    }),
  );
  await waitUntilTableExists({ client }, { TableName: NAMES.tableName });
});
```

```

    }},
    new ScenarioOutput(
      "createdTable",
      MESSAGES.createdTable.replace("${TABLE_NAME}", NAMES.tableName),
    ),
    new ScenarioOutput(
      "populatingTable",
      MESSAGES.populatingTable.replace("${TABLE_NAME}", NAMES.tableName),
    ),
    new ScenarioAction("populateTable", () => {
      const client = new DynamoDBClient({});
      /**
       * @type {{ default: import("@aws-sdk/client-dynamodb").PutRequest['Item'][] }}
       */
      const recommendations = JSON.parse(
        readFileSync(join(RESOURCES_PATH, "recommendations.json")),
      );

      return client.send(
        new BatchWriteItemCommand({
          RequestItems: {
            [NAMES.tableName]: recommendations.map((item) => ({
              PutRequest: { Item: item },
            })),
          },
        }),
      );
    }),
    new ScenarioOutput(
      "populatedTable",
      MESSAGES.populatedTable.replace("${TABLE_NAME}", NAMES.tableName),
    ),
    new ScenarioOutput(
      "creatingKeyPair",
      MESSAGES.creatingKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
    ),
    new ScenarioAction("createKeyPair", async () => {
      const client = new EC2Client({});
      const { KeyMaterial } = await client.send(
        new CreateKeyPairCommand({
          KeyName: NAMES.keyPairName,
        }),
      );
    });

```



```

    writeFileSync(`${NAMES.keyPairName}.pem`, KeyMaterial, { mode: 0o600 });
  }),
  new ScenarioOutput(
    "createdKeyPair",
    MESSAGES.createdKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
  ),
  new ScenarioOutput(
    "creatingInstancePolicy",
    MESSAGES.creatingInstancePolicy.replace(
      "${INSTANCE_POLICY_NAME}",
      NAMES.instancePolicyName,
    ),
  ),
  new ScenarioAction("createInstancePolicy", async (state) => {
    const client = new IAMClient({});
    const {
      Policy: { Arn },
    } = await client.send(
      new CreatePolicyCommand({
        PolicyName: NAMES.instancePolicyName,
        PolicyDocument: readFileSync(
          join(RESOURCES_PATH, "instance_policy.json"),
        ),
      }),
    );
    state.instancePolicyArn = Arn;
  }),
  new ScenarioOutput("createdInstancePolicy", (state) =>
    MESSAGES.createdInstancePolicy
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
      .replace("${INSTANCE_POLICY_ARN}", state.instancePolicyArn),
  ),
  new ScenarioOutput(
    "creatingInstanceRole",
    MESSAGES.creatingInstanceRole.replace(
      "${INSTANCE_ROLE_NAME}",
      NAMES.instanceRoleName,
    ),
  ),
  new ScenarioAction("createInstanceRole", () => {
    const client = new IAMClient({});
    return client.send(
      new CreateRoleCommand({
        RoleName: NAMES.instanceRoleName,

```

```

        AssumeRolePolicyDocument: readFileSync(
            join(ROOT, "assume-role-policy.json"),
        ),
    }},
);
}},
new ScenarioOutput(
    "createdInstanceRole",
    MESSAGES.createdInstanceRole.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
    ),
),
new ScenarioOutput(
    "attachingPolicyToRole",
    MESSAGES.attachingPolicyToRole
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName)
        .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName),
),
new ScenarioAction("attachPolicyToRole", async (state) => {
    const client = new IAMClient({});
    await client.send(
        new AttachRolePolicyCommand({
            RoleName: NAMES.instanceRoleName,
            PolicyArn: state.instancePolicyArn,
        }),
    );
}),
new ScenarioOutput(
    "attachedPolicyToRole",
    MESSAGES.attachedPolicyToRole
        .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
),
new ScenarioOutput(
    "creatingInstanceProfile",
    MESSAGES.creatingInstanceProfile.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.instanceProfileName,
    ),
),
new ScenarioAction("createInstanceProfile", async (state) => {
    const client = new IAMClient({});
    const {

```

```

    InstanceProfile: { Arn },
  } = await client.send(
    new CreateInstanceProfileCommand({
      InstanceProfileName: NAMES.instanceProfileName,
    }),
  );
  state.instanceProfileArn = Arn;

  await waitUntilInstanceProfileExists(
    { client },
    { InstanceProfileName: NAMES.instanceProfileName },
  );
}),
new ScenarioOutput("createdInstanceProfile", (state) =>
  MESSAGES.createdInstanceProfile
    .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
    .replace("${INSTANCE_PROFILE_ARN}", state.instanceProfileArn),
),
new ScenarioOutput(
  "addingRoleToInstanceProfile",
  MESSAGES.addingRoleToInstanceProfile
    .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
    .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
),
new ScenarioAction("addRoleToInstanceProfile", () => {
  const client = new IAMClient({});
  return client.send(
    new AddRoleToInstanceProfileCommand({
      RoleName: NAMES.instanceRoleName,
      InstanceProfileName: NAMES.instanceProfileName,
    }),
  );
}),
new ScenarioOutput(
  "addedRoleToInstanceProfile",
  MESSAGES.addedRoleToInstanceProfile
    .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
    .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
),
...initParamsSteps,
new ScenarioOutput("creatingLaunchTemplate", MESSAGES.creatingLaunchTemplate),
new ScenarioAction("createLaunchTemplate", async () => {
  const ssmClient = new SSMClient({});
  const { Parameter } = await ssmClient.send(

```

```

    new GetParameterCommand({
      Name: "/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2",
    }),
  );
const ec2Client = new EC2Client({});
await ec2Client.send(
  new CreateLaunchTemplateCommand({
    LaunchTemplateName: NAMES.launchTemplateName,
    LaunchTemplateData: {
      InstanceType: "t3.micro",
      ImageId: Parameter.Value,
      IamInstanceProfile: { Name: NAMES.instanceProfileName },
      UserData: readFileSync(
        join(RESOURCES_PATH, "server_startup_script.sh"),
      ).toString("base64"),
      KeyName: NAMES.keyPairName,
    },
  }),
);
}),
new ScenarioOutput(
  "createdLaunchTemplate",
  MESSAGES.createdLaunchTemplate.replace(
    "${LAUNCH_TEMPLATE_NAME}",
    NAMES.launchTemplateName,
  ),
),
new ScenarioOutput(
  "creatingAutoScalingGroup",
  MESSAGES.creatingAutoScalingGroup.replace(
    "${AUTO_SCALING_GROUP_NAME}",
    NAMES.autoScalingGroupName,
  ),
),
new ScenarioAction("createAutoScalingGroup", async (state) => {
  const ec2Client = new EC2Client({});
  const { AvailabilityZones } = await ec2Client.send(
    new DescribeAvailabilityZonesCommand({}),
  );
  state.availabilityZoneNames = AvailabilityZones.map((az) => az.ZoneName);
  const autoScalingClient = new AutoScalingClient({});
  await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
    autoScalingClient.send(
      new CreateAutoScalingGroupCommand({

```

```

        AvailabilityZones: state.availabilityZoneNames,
        AutoScalingGroupName: NAMES.autoScalingGroupName,
        LaunchTemplate: {
            LaunchTemplateName: NAMES.launchTemplateName,
            Version: "$Default",
        },
        MinSize: 3,
        MaxSize: 3,
    )),
),
);
}},
new ScenarioOutput(
    "createdAutoScalingGroup",
    /**
     * @param {{ availabilityZoneNames: string[] }} state
     */
    (state) =>
        MESSAGES.createdAutoScalingGroup
            .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName)
            .replace(
                "${AVAILABILITY_ZONE_NAMES}",
                state.availabilityZoneNames.join(", "),
            ),
),
new ScenarioInput("confirmContinue", MESSAGES.confirmContinue, {
    type: "confirm",
}),
new ScenarioOutput("loadBalancer", MESSAGES.loadBalancer),
new ScenarioOutput("gettingVpc", MESSAGES.gettingVpc),
new ScenarioAction("getVpc", async (state) => {
    const client = new EC2Client({});
    const { Vpcs } = await client.send(
        new DescribeVpcsCommand({
            Filters: [{ Name: "is-default", Values: ["true"] }],
        }),
    );
    state.defaultVpc = Vpcs[0].VpcId;
}),
new ScenarioOutput("gotVpc", (state) =>
    MESSAGES.gotVpc.replace("${VPC_ID}", state.defaultVpc),
),
new ScenarioOutput("gettingSubnets", MESSAGES.gettingSubnets),
new ScenarioAction("getSubnets", async (state) => {

```

```

const client = new EC2Client({});
const { Subnets } = await client.send(
  new DescribeSubnetsCommand({
    Filters: [
      { Name: "vpc-id", Values: [state.defaultVpc] },
      { Name: "availability-zone", Values: state.availabilityZoneNames },
      { Name: "default-for-az", Values: ["true"] },
    ],
  }),
);
state.subnets = Subnets.map((subnet) => subnet.SubnetId);
}),
new ScenarioOutput(
  "gotSubnets",
  /**
   * @param {{ subnets: string[] }} state
   */
  (state) =>
    MESSAGES.gotSubnets.replace("${SUBNETS}", state.subnets.join(", ")),
),
new ScenarioOutput(
  "creatingLoadBalancerTargetGroup",
  MESSAGES.creatingLoadBalancerTargetGroup.replace(
    "${TARGET_GROUP_NAME}",
    NAMES.loadBalancerTargetGroupName,
  ),
),
new ScenarioAction("createLoadBalancerTargetGroup", async (state) => {
  const client = new ElasticLoadBalancingV2Client({});
  const { TargetGroups } = await client.send(
    new CreateTargetGroupCommand({
      Name: NAMES.loadBalancerTargetGroupName,
      Protocol: "HTTP",
      Port: 80,
      HealthCheckPath: "/healthcheck",
      HealthCheckIntervalSeconds: 10,
      HealthCheckTimeoutSeconds: 5,
      HealthyThresholdCount: 2,
      UnhealthyThresholdCount: 2,
      VpcId: state.defaultVpc,
    }),
  );
  const targetGroup = TargetGroups[0];
  state.targetGroupArn = targetGroup.TargetGroupArn;
});

```

```

    state.targetGroupProtocol = targetGroup.Protocol;
    state.targetGroupPort = targetGroup.Port;
  )),
  new ScenarioOutput(
    "createdLoadBalancerTargetGroup",
    MESSAGES.createdLoadBalancerTargetGroup.replace(
      "${TARGET_GROUP_NAME}",
      NAMES.loadBalancerTargetGroupName,
    ),
  ),
  new ScenarioOutput(
    "creatingLoadBalancer",
    MESSAGES.creatingLoadBalancer.replace("${LB_NAME}", NAMES.loadBalancerName),
  ),
  new ScenarioAction("createLoadBalancer", async (state) => {
    const client = new ElasticLoadBalancingV2Client({});
    const { LoadBalancers } = await client.send(
      new CreateLoadBalancerCommand({
        Name: NAMES.loadBalancerName,
        Subnets: state.subnets,
      }),
    );
    state.loadBalancerDns = LoadBalancers[0].DNSName;
    state.loadBalancerArn = LoadBalancers[0].LoadBalancerArn;
    await waitUntilLoadBalancerAvailable(
      { client },
      { Names: [NAMES.loadBalancerName] },
    );
  )),
  new ScenarioOutput("createdLoadBalancer", (state) =>
    MESSAGES.createdLoadBalancer
      .replace("${LB_NAME}", NAMES.loadBalancerName)
      .replace("${DNS_NAME}", state.loadBalancerDns),
  ),
  new ScenarioOutput(
    "creatingListener",
    MESSAGES.creatingLoadBalancerListener
      .replace("${LB_NAME}", NAMES.loadBalancerName)
      .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName),
  ),
  new ScenarioAction("createListener", async (state) => {
    const client = new ElasticLoadBalancingV2Client({});
    const { Listeners } = await client.send(
      new CreateListenerCommand({

```

```

        LoadBalancerArn: state.loadBalancerArn,
        Protocol: state.targetGroupProtocol,
        Port: state.targetGroupPort,
        DefaultActions: [
            { Type: "forward", TargetGroupArn: state.targetGroupArn },
        ],
    })),
);
const listener = Listeners[0];
state.loadBalancerListenerArn = listener.ListenerArn;
}),
new ScenarioOutput("createdListener", (state) =>
    MESSAGES.createdLoadBalancerListener.replace(
        "${LB_LISTENER_ARN}",
        state.loadBalancerListenerArn,
    ),
),
new ScenarioOutput(
    "attachingLoadBalancerTargetGroup",
    MESSAGES.attachingLoadBalancerTargetGroup
        .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName)
        .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName),
),
new ScenarioAction("attachLoadBalancerTargetGroup", async (state) => {
    const client = new AutoScalingClient({});
    await client.send(
        new AttachLoadBalancerTargetGroupsCommand({
            AutoScalingGroupName: NAMES.autoScalingGroupName,
            TargetGroupARNs: [state.targetGroupArn],
        }),
    );
}),
new ScenarioOutput(
    "attachedLoadBalancerTargetGroup",
    MESSAGES.attachedLoadBalancerTargetGroup,
),
new ScenarioOutput("verifyingInboundPort", MESSAGES.verifyingInboundPort),
new ScenarioAction(
    "verifyInboundPort",
    /**
     *
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-ec2').SecurityGroup}}
state
    */

```



```

async (state) => {
  const client = new EC2Client({});
  const { SecurityGroups } = await client.send(
    new DescribeSecurityGroupsCommand({
      Filters: [{ Name: "group-name", Values: ["default"] }],
    }),
  );
  if (!SecurityGroups) {
    state.verifyInboundPortError = new Error(MESSAGES.noSecurityGroups);
  }
  state.defaultSecurityGroup = SecurityGroups[0];

  /**
   * @type {string}
   */
  const ipResponse = (await axios.get("http://checkip.amazonaws.com")).data;
  state.myIp = ipResponse.trim();
  const myIpRules = state.defaultSecurityGroup.IpPermissions.filter(
    ({ IpRanges }) =>
      IpRanges.some(
        ({ CidrIp }) =>
          CidrIp.startsWith(state.myIp) || CidrIp === "0.0.0.0/0",
      ),
  )
    .filter(({ IpProtocol }) => IpProtocol === "tcp")
    .filter(({ FromPort }) => FromPort === 80);

  state.myIpRules = myIpRules;
},
),
new ScenarioOutput(
  "verifiedInboundPort",
  /**
   * @param {{ myIpRules: any[] }} state
   */
  (state) => {
    if (state.myIpRules.length > 0) {
      return MESSAGES.foundIpRules.replace(
        "${IP_RULES}",
        JSON.stringify(state.myIpRules, null, 2),
      );
    }
    return MESSAGES.noIpRules;
  },
),

```

```

    ),
    new ScenarioInput(
      "shouldAddInboundRule",
      /**
       * @param {{ myIpRules: any[] }} state
       */
      (state) => {
        if (state.myIpRules.length > 0) {
          return false;
        }
        return MESSAGES.noIpRules;
      },
      { type: "confirm" },
    ),
    new ScenarioAction(
      "addInboundRule",
      /**
       * @param {{ defaultSecurityGroup: import('@aws-sdk/client-ec2').SecurityGroup }} state
       */
      async (state) => {
        if (!state.shouldAddInboundRule) {
          return;
        }

        const client = new EC2Client({});
        await client.send(
          new AuthorizeSecurityGroupIngressCommand({
            GroupId: state.defaultSecurityGroup.GroupId,
            CidrIp: `${state.myIp}/32`,
            FromPort: 80,
            ToPort: 80,
            IpProtocol: "tcp",
          }),
        );
      },
    ),
    new ScenarioOutput("addedInboundRule", (state) => {
      if (state.shouldAddInboundRule) {
        return MESSAGES.addedInboundRule.replace("${IP_ADDRESS}", state.myIp);
      }
      return false;
    }),
    new ScenarioOutput("verifyingEndpoint", (state) =>

```

```

    MESSAGES.verifyingEndpoint.replace("${DNS_NAME}", state.loadBalancerDns),
  ),
  new ScenarioAction("verifyEndpoint", async (state) => {
    try {
      const response = await retry({ intervalInMs: 2000, maxRetries: 30 }, () =>
        axios.get(`http://${state.loadBalancerDns}`),
      );
      state.endpointResponse = JSON.stringify(response.data, null, 2);
    } catch (e) {
      state.verifyEndpointError = e;
    }
  }),
  new ScenarioOutput("verifiedEndpoint", (state) => {
    if (state.verifyEndpointError) {
      console.error(state.verifyEndpointError);
    } else {
      return MESSAGES.verifiedEndpoint.replace(
        "${ENDPOINT_RESPONSE}",
        state.endpointResponse,
      );
    }
  }),
  saveState,
];

```

Erstellen Sie Schritte, um die Demo auszuführen.

```

import { readFileSync } from "node:fs";
import { join } from "node:path";

import axios from "axios";

import {
  DescribeTargetGroupsCommand,
  DescribeTargetHealthCommand,
  ElasticLoadBalancingV2Client,
} from "@aws-sdk/client-elastic-load-balancing-v2";
import {
  DescribeInstanceInformationCommand,
  PutParameterCommand,
  SSMClient,
  SendCommandCommand,

```

```
} from "@aws-sdk/client-ssm";
import {
  IAMClient,
  CreatePolicyCommand,
  CreateRoleCommand,
  AttachRolePolicyCommand,
  CreateInstanceProfileCommand,
  AddRoleToInstanceProfileCommand,
  waitUntilInstanceProfileExists,
} from "@aws-sdk/client-iam";
import {
  AutoScalingClient,
  DescribeAutoScalingGroupsCommand,
  TerminateInstanceInAutoScalingGroupCommand,
} from "@aws-sdk/client-auto-scaling";
import {
  DescribeIamInstanceProfileAssociationsCommand,
  EC2Client,
  RebootInstancesCommand,
  ReplaceIamInstanceProfileAssociationCommand,
} from "@aws-sdk/client-ec2";

import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/scenario.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES, RESOURCES_PATH } from "../constants.js";
import { findLoadBalancer } from "../shared.js";

const getRecommendation = new ScenarioAction(
  "getRecommendation",
  async (state) => {
    const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
    if (loadBalancer) {
      state.loadBalancerDnsName = loadBalancer.DNSName;
      try {
        state.recommendation = (
          await axios.get(`http://${state.loadBalancerDnsName}`)
        ).data;
      } catch (e) {
        state.recommendation = e instanceof Error ? e.message : e;
      }
    }
  }
);
```

```

    }
  } else {
    throw new Error(MESSAGES.demoFindLoadBalancerError);
  }
},
);

const getRecommendationResult = new ScenarioOutput(
  "getRecommendationResult",
  (state) =>
    `Recommendation:\n${JSON.stringify(state.recommendation, null, 2)}`,
  { preformatted: true },
);

const getHealthCheck = new ScenarioAction("getHealthCheck", async (state) => {
  const client = new ElasticLoadBalancingV2Client({});
  const { TargetGroups } = await client.send(
    new DescribeTargetGroupsCommand({
      Names: [NAMES.loadBalancerTargetGroupName],
    }),
  );
  const { TargetHealthDescriptions } = await client.send(
    new DescribeTargetHealthCommand({
      TargetGroupArn: TargetGroups[0].TargetGroupArn,
    }),
  );
  state.targetHealthDescriptions = TargetHealthDescriptions;
});

const getHealthCheckResult = new ScenarioOutput(
  "getHealthCheckResult",
  /**
   * @param {{ targetHealthDescriptions: import('@aws-sdk/client-elastic-load-balancing-v2').TargetHealthDescription[] }} state
   */
  (state) => {
    const status = state.targetHealthDescriptions
      .map((th) => `${th.Target.Id}: ${th.TargetHealth.State}`)
      .join("\n");
    return `Health check:\n${status}`;
  },
  { preformatted: true },
);

```

```
const loadBalancerLoop = new ScenarioAction(
  "loadBalancerLoop",
  getRecommendation.action,
  {
    whileConfig: {
      whileFn: ({ loadBalancerCheck }) => loadBalancerCheck,
      input: new ScenarioInput(
        "loadBalancerCheck",
        MESSAGES.demoLoadBalancerCheck,
        {
          type: "confirm",
        },
      ),
      output: getRecommendationResult,
    },
  },
);

const healthCheckLoop = new ScenarioAction(
  "healthCheckLoop",
  getHealthCheck.action,
  {
    whileConfig: {
      whileFn: ({ healthCheck }) => healthCheck,
      input: new ScenarioInput("healthCheck", MESSAGES.demoHealthCheck, {
        type: "confirm",
      }),
      output: getHealthCheckResult,
    },
  },
);

const statusSteps = [
  getRecommendation,
  getRecommendationResult,
  getHealthCheck,
  getHealthCheckResult,
];

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const demoSteps = [
```

```

new ScenarioOutput("header", MESSAGES.demoHeader, { header: true }),
new ScenarioOutput("sanityCheck", MESSAGES.demoSanityCheck),
...statusSteps,
new ScenarioInput(
  "brokenDependencyConfirmation",
  MESSAGES.demoBrokenDependencyConfirmation,
  { type: "confirm" },
),
new ScenarioAction("brokenDependency", async (state) => {
  if (!state.brokenDependencyConfirmation) {
    process.exit();
  } else {
    const client = new SSMClient({});
    state.badTableName = `fake-table-${Date.now()}`;
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: state.badTableName,
        Overwrite: true,
        Type: "String",
      }),
    );
  }
}),
new ScenarioOutput("testBrokenDependency", (state) =>
  MESSAGES.demoTestBrokenDependency.replace(
    "${TABLE_NAME}",
    state.badTableName,
  ),
),
...statusSteps,
new ScenarioInput(
  "staticResponseConfirmation",
  MESSAGES.demoStaticResponseConfirmation,
  { type: "confirm" },
),
new ScenarioAction("staticResponse", async (state) => {
  if (!state.staticResponseConfirmation) {
    process.exit();
  } else {
    const client = new SSMClient({});
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmFailureResponseKey,

```

```

        Value: "static",
        Overwrite: true,
        Type: "String",
    })),
    );
}
}),
new ScenarioOutput("testStaticResponse", MESSAGES.demoTestStaticResponse),
...statusSteps,
new ScenarioInput(
    "badCredentialsConfirmation",
    MESSAGES.demoBadCredentialsConfirmation,
    { type: "confirm" },
),
new ScenarioAction("badCredentialsExit", (state) => {
    if (!state.badCredentialsConfirmation) {
        process.exit();
    }
}),
new ScenarioAction("fixDynamoDBName", async () => {
    const client = new SSMClient({});
    await client.send(
        new PutParameterCommand({
            Name: NAMES.ssmTableNameKey,
            Value: NAMES.tableName,
            Overwrite: true,
            Type: "String",
        })
    );
}),
new ScenarioAction(
    "badCredentials",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-auto-scaling').Instance }}
state
    */
    async (state) => {
        await createSsmOnlyInstanceProfile();
        const autoScalingClient = new AutoScalingClient({});
        const { AutoScalingGroups } = await autoScalingClient.send(
            new DescribeAutoScalingGroupsCommand({
                AutoScalingGroupNames: [NAMES.autoScalingGroupName],
            })
        );
    }
);

```



```
state.targetInstance = AutoScalingGroups[0].Instances[0];
const ec2Client = new EC2Client({});
const { IamInstanceProfileAssociations } = await ec2Client.send(
  new DescribeIamInstanceProfileAssociationsCommand({
    Filters: [
      { Name: "instance-id", Values: [state.targetInstance.InstanceId] },
    ],
  }),
);
state.instanceProfileAssociationId =
  IamInstanceProfileAssociations[0].AssociationId;
await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
  ec2Client.send(
    new ReplaceIamInstanceProfileAssociationCommand({
      AssociationId: state.instanceProfileAssociationId,
      IamInstanceProfile: { Name: NAMES.ssmOnlyInstanceProfileName },
    }),
  ),
);

await ec2Client.send(
  new RebootInstancesCommand({
    InstanceIds: [state.targetInstance.InstanceId],
  }),
);

const ssmClient = new SSMClient({});
await retry({ intervalInMs: 20000, maxRetries: 15 }, async () => {
  const { InstanceInformationList } = await ssmClient.send(
    new DescribeInstanceInformationCommand({}),
  );

  const instance = InstanceInformationList.find(
    (info) => info.InstanceId === state.targetInstance.InstanceId,
  );

  if (!instance) {
    throw new Error("Instance not found.");
  }
});

await ssmClient.send(
  new SendCommandCommand({
    InstanceIds: [state.targetInstance.InstanceId],
```

```

        DocumentName: "AWS-RunShellScript",
        Parameters: { commands: ["cd / && sudo python3 server.py 80"] },
    })),
    );
},
),
new ScenarioOutput(
    "testBadCredentials",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-ssm').InstanceInformation}}
state
    */
    (state) =>
        MESSAGES.demoTestBadCredentials.replace(
            "${INSTANCE_ID}",
            state.targetInstance.InstanceId,
        ),
    ),
loadBalancerLoop,
new ScenarioInput(
    "deepHealthCheckConfirmation",
    MESSAGES.demoDeepHealthCheckConfirmation,
    { type: "confirm" },
),
new ScenarioAction("deepHealthCheckExit", (state) => {
    if (!state.deepHealthCheckConfirmation) {
        process.exit();
    }
}),
new ScenarioAction("deepHealthCheck", async () => {
    const client = new SSMClient({});
    await client.send(
        new PutParameterCommand({
            Name: NAMES.ssmHealthCheckKey,
            Value: "deep",
            Overwrite: true,
            Type: "String",
        }),
    );
}),
new ScenarioOutput("testDeepHealthCheck", MESSAGES.demoTestDeepHealthCheck),
healthCheckLoop,
loadBalancerLoop,
new ScenarioInput(

```

```

    "killInstanceConfirmation",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-
    ssm').InstanceInformation }} state
     */
    (state) =>
        MESSAGES.demoKillInstanceConfirmation.replace(
            "${INSTANCE_ID}",
            state.targetInstance.InstanceId,
        ),
    { type: "confirm" },
),
new ScenarioAction("killInstanceExit", (state) => {
    if (!state.killInstanceConfirmation) {
        process.exit();
    }
}),
new ScenarioAction(
    "killInstance",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-
    ssm').InstanceInformation }} state
     */
    async (state) => {
        const client = new AutoScalingClient({});
        await client.send(
            new TerminateInstanceInAutoScalingGroupCommand({
                InstanceId: state.targetInstance.InstanceId,
                ShouldDecrementDesiredCapacity: false,
            }),
        );
    },
),
new ScenarioOutput("testKillInstance", MESSAGES.demoTestKillInstance),
healthCheckLoop,
loadBalancerLoop,
new ScenarioInput("failOpenConfirmation", MESSAGES.demoFailOpenConfirmation, {
    type: "confirm",
}),
new ScenarioAction("failOpenExit", (state) => {
    if (!state.failOpenConfirmation) {
        process.exit();
    }
}),
}),

```

```

new ScenarioAction("failOpen", () => {
  const client = new SSMClient({});
  return client.send(
    new PutParameterCommand({
      Name: NAMES.ssmTableNameKey,
      Value: `fake-table-${Date.now()}`,
      Overwrite: true,
      Type: "String",
    }),
  );
}),
new ScenarioOutput("testFailOpen", MESSAGES.demoFailOpenTest),
healthCheckLoop,
loadBalancerLoop,
new ScenarioInput(
  "resetTableConfirmation",
  MESSAGES.demoResetTableConfirmation,
  { type: "confirm" },
),
new ScenarioAction("resetTableExit", (state) => {
  if (!state.resetTableConfirmation) {
    process.exit();
  }
}),
new ScenarioAction("resetTable", async () => {
  const client = new SSMClient({});
  await client.send(
    new PutParameterCommand({
      Name: NAMES.ssmTableNameKey,
      Value: NAMES.tableName,
      Overwrite: true,
      Type: "String",
    }),
  );
}),
new ScenarioOutput("testResetTable", MESSAGES.demoTestResetTable),
healthCheckLoop,
loadBalancerLoop,
];

async function createSsmOnlyInstanceProfile() {
  const iamClient = new IAMClient({});
  const { Policy } = await iamClient.send(
    new CreatePolicyCommand({

```

```
    PolicyName: NAMES.ssmOnlyPolicyName,
    PolicyDocument: readFileSync(
      join(RESOURCES_PATH, "ssm_only_policy.json"),
    ),
  )),
);
await iamClient.send(
  new CreateRoleCommand({
    RoleName: NAMES.ssmOnlyRoleName,
    AssumeRolePolicyDocument: JSON.stringify({
      Version: "2012-10-17",
      Statement: [
        {
          Effect: "Allow",
          Principal: { Service: "ec2.amazonaws.com" },
          Action: "sts:AssumeRole",
        },
      ],
    }),
  )),
);
await iamClient.send(
  new AttachRolePolicyCommand({
    RoleName: NAMES.ssmOnlyRoleName,
    PolicyArn: Policy.Arn,
  )),
);
await iamClient.send(
  new AttachRolePolicyCommand({
    RoleName: NAMES.ssmOnlyRoleName,
    PolicyArn: "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore",
  )),
);
const { InstanceProfile } = await iamClient.send(
  new CreateInstanceProfileCommand({
    InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
  )),
);
await waitUntilInstanceProfileExists(
  { client: iamClient },
  { InstanceProfileName: NAMES.ssmOnlyInstanceProfileName },
);
await iamClient.send(
  new AddRoleToInstanceProfileCommand({
```

```
        InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,  
        RoleName: NAMES.ssmOnlyRoleName,  
    }},  
    );  
  
    return InstanceProfile;  
}
```

Erstellen Sie Schritte, um alle Ressourcen zu vernichten.

```
import { unlinkSync } from "node:fs";  
  
import { DynamoDBClient, DeleteTableCommand } from "@aws-sdk/client-dynamodb";  
import {  
    EC2Client,  
    DeleteKeyPairCommand,  
    DeleteLaunchTemplateCommand,  
    RevokeSecurityGroupIngressCommand,  
} from "@aws-sdk/client-ec2";  
import {  
    IAMClient,  
    DeleteInstanceProfileCommand,  
    RemoveRoleFromInstanceProfileCommand,  
    DeletePolicyCommand,  
    DeleteRoleCommand,  
    DetachRolePolicyCommand,  
    paginateListPolicies,  
} from "@aws-sdk/client-iam";  
import {  
    AutoScalingClient,  
    DeleteAutoScalingGroupCommand,  
    TerminateInstanceInAutoScalingGroupCommand,  
    UpdateAutoScalingGroupCommand,  
    paginateDescribeAutoScalingGroups,  
} from "@aws-sdk/client-auto-scaling";  
import {  
    DeleteLoadBalancerCommand,  
    DeleteTargetGroupCommand,  
    DescribeTargetGroupsCommand,  
    ElasticLoadBalancingV2Client,  
} from "@aws-sdk/client-elastic-load-balancing-v2";
```

```

import {
  ScenarioOutput,
  ScenarioInput,
  ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { loadState } from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES } from "../constants.js";
import { findLoadBalancer } from "../shared.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const destroySteps = [
  loadState,
  new ScenarioInput("destroy", MESSAGES.destroy, { type: "confirm" }),
  new ScenarioAction(
    "abort",
    (state) => state.destroy === false && process.exit(),
  ),
  new ScenarioAction("deleteTable", async (c) => {
    try {
      const client = new DynamoDBClient({});
      await client.send(new DeleteTableCommand({ TableName: NAMES.tableName }));
    } catch (e) {
      c.deleteTableError = e;
    }
  }),
  new ScenarioOutput("deleteTableResult", (state) => {
    if (state.deleteTableError) {
      console.error(state.deleteTableError);
      return MESSAGES.deleteTableError.replace(
        "${TABLE_NAME}",
        NAMES.tableName,
      );
    }
    return MESSAGES.deletedTable.replace("${TABLE_NAME}", NAMES.tableName);
  }),
  new ScenarioAction("deleteKeyPair", async (state) => {
    try {
      const client = new EC2Client({});
      await client.send(
        new DeleteKeyPairCommand({ KeyName: NAMES.keyPairName }),
      ),
    }
  )
];

```

```

    );
    unlinkSync(`${NAMES.keyPairName}.pem`);
  } catch (e) {
    state.deleteKeyPairError = e;
  }
}),
new ScenarioOutput("deleteKeyPairResult", (state) => {
  if (state.deleteKeyPairError) {
    console.error(state.deleteKeyPairError);
    return MESSAGES.deleteKeyPairError.replace(
      "${KEY_PAIR_NAME}",
      NAMES.keyPairName,
    );
  }
  return MESSAGES.deletedKeyPair.replace(
    "${KEY_PAIR_NAME}",
    NAMES.keyPairName,
  );
}),
new ScenarioAction("detachPolicyFromRole", async (state) => {
  try {
    const client = new IAMClient({});
    const policy = await findPolicy(NAMES.instancePolicyName);

    if (!policy) {
      state.detachPolicyFromRoleError = new Error(
        `Policy ${NAMES.instancePolicyName} not found.`
      );
    } else {
      await client.send(
        new DetachRolePolicyCommand({
          RoleName: NAMES.instanceRoleName,
          PolicyArn: policy.Arn,
        }),
      );
    }
  } catch (e) {
    state.detachPolicyFromRoleError = e;
  }
}),
new ScenarioOutput("detachedPolicyFromRole", (state) => {
  if (state.detachPolicyFromRoleError) {
    console.error(state.detachPolicyFromRoleError);
    return MESSAGES.detachPolicyFromRoleError

```



```

        .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    }
    return MESSAGES.detachedPolicyFromRole
        .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    })),
    new ScenarioAction("deleteInstancePolicy", async (state) => {
        const client = new IAMClient({});
        const policy = await findPolicy(NAMES.instancePolicyName);

        if (!policy) {
            state.deletePolicyError = new Error(
                `Policy ${NAMES.instancePolicyName} not found.`
            );
        } else {
            return client.send(
                new DeletePolicyCommand({
                    PolicyArn: policy.Arn,
                })
            );
        }
    })),
    new ScenarioOutput("deletePolicyResult", (state) => {
        if (state.deletePolicyError) {
            console.error(state.deletePolicyError);
            return MESSAGES.deletePolicyError.replace(
                "${INSTANCE_POLICY_NAME}",
                NAMES.instancePolicyName,
            );
        }
        return MESSAGES.deletedPolicy.replace(
            "${INSTANCE_POLICY_NAME}",
            NAMES.instancePolicyName,
        );
    })),
    new ScenarioAction("removeRoleFromInstanceProfile", async (state) => {
        try {
            const client = new IAMClient({});
            await client.send(
                new RemoveRoleFromInstanceProfileCommand({
                    RoleName: NAMES.instanceRoleName,
                    InstanceProfileName: NAMES.instanceProfileName,
                })
            );
        }
    })),

```

```
    );
  } catch (e) {
    state.removeRoleFromInstanceProfileError = e;
  }
}),
new ScenarioOutput("removeRoleFromInstanceProfileResult", (state) => {
  if (state.removeRoleFromInstanceProfile) {
    console.error(state.removeRoleFromInstanceProfileError);
    return MESSAGES.removeRoleFromInstanceProfileError
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
  }
  return MESSAGES.removedRoleFromInstanceProfile
    .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
    .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
}),
new ScenarioAction("deleteInstanceRole", async (state) => {
  try {
    const client = new IAMClient({});
    await client.send(
      new DeleteRoleCommand({
        RoleName: NAMES.instanceRoleName,
      }),
    );
  } catch (e) {
    state.deleteInstanceRoleError = e;
  }
}),
new ScenarioOutput("deleteInstanceRoleResult", (state) => {
  if (state.deleteInstanceRoleError) {
    console.error(state.deleteInstanceRoleError);
    return MESSAGES.deleteInstanceRoleError.replace(
      "${INSTANCE_ROLE_NAME}",
      NAMES.instanceRoleName,
    );
  }
  return MESSAGES.deletedInstanceRole.replace(
    "${INSTANCE_ROLE_NAME}",
    NAMES.instanceRoleName,
  );
}),
new ScenarioAction("deleteInstanceProfile", async (state) => {
  try {
    const client = new IAMClient({});
```

```
    await client.send(
      new DeleteInstanceProfileCommand({
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
  } catch (e) {
    state.deleteInstanceProfileError = e;
  }
}),
new ScenarioOutput("deleteInstanceProfileResult", (state) => {
  if (state.deleteInstanceProfileError) {
    console.error(state.deleteInstanceProfileError);
    return MESSAGES.deleteInstanceProfileError.replace(
      "${INSTANCE_PROFILE_NAME}",
      NAMES.instanceProfileName,
    );
  }
  return MESSAGES.deletedInstanceProfile.replace(
    "${INSTANCE_PROFILE_NAME}",
    NAMES.instanceProfileName,
  );
}),
new ScenarioAction("deleteAutoScalingGroup", async (state) => {
  try {
    await terminateGroupInstances(NAMES.autoScalingGroupName);
    await retry({ intervalInMs: 60000, maxRetries: 60 }, async () => {
      await deleteAutoScalingGroup(NAMES.autoScalingGroupName);
    });
  } catch (e) {
    state.deleteAutoScalingGroupError = e;
  }
}),
new ScenarioOutput("deleteAutoScalingGroupResult", (state) => {
  if (state.deleteAutoScalingGroupError) {
    console.error(state.deleteAutoScalingGroupError);
    return MESSAGES.deleteAutoScalingGroupError.replace(
      "${AUTO_SCALING_GROUP_NAME}",
      NAMES.autoScalingGroupName,
    );
  }
  return MESSAGES.deletedAutoScalingGroup.replace(
    "${AUTO_SCALING_GROUP_NAME}",
    NAMES.autoScalingGroupName,
  );
});
```

```
    }},
    new ScenarioAction("deleteLaunchTemplate", async (state) => {
      const client = new EC2Client({});
      try {
        await client.send(
          new DeleteLaunchTemplateCommand({
            LaunchTemplateName: NAMES.launchTemplateName,
          }),
        );
      } catch (e) {
        state.deleteLaunchTemplateError = e;
      }
    })),
    new ScenarioOutput("deleteLaunchTemplateResult", (state) => {
      if (state.deleteLaunchTemplateError) {
        console.error(state.deleteLaunchTemplateError);
        return MESSAGES.deleteLaunchTemplateError.replace(
          "${LAUNCH_TEMPLATE_NAME}",
          NAMES.launchTemplateName,
        );
      }
      return MESSAGES.deletedLaunchTemplate.replace(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
      );
    })),
    new ScenarioAction("deleteLoadBalancer", async (state) => {
      try {
        const client = new ElasticLoadBalancingV2Client({});
        const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
        await client.send(
          new DeleteLoadBalancerCommand({
            LoadBalancerArn: loadBalancer.LoadBalancerArn,
          }),
        );
        await retry({ intervalInMs: 1000, maxRetries: 60 }, async () => {
          const lb = await findLoadBalancer(NAMES.loadBalancerName);
          if (lb) {
            throw new Error("Load balancer still exists.");
          }
        });
      } catch (e) {
        state.deleteLoadBalancerError = e;
      }
    })
```

```
    }},
    new ScenarioOutput("deleteLoadBalancerResult", (state) => {
      if (state.deleteLoadBalancerError) {
        console.error(state.deleteLoadBalancerError);
        return MESSAGES.deleteLoadBalancerError.replace(
          "${LB_NAME}",
          NAMES.loadBalancerName,
        );
      }
      return MESSAGES.deletedLoadBalancer.replace(
        "${LB_NAME}",
        NAMES.loadBalancerName,
      );
    }),
    new ScenarioAction("deleteLoadBalancerTargetGroup", async (state) => {
      const client = new ElasticLoadBalancingV2Client({});
      try {
        const { TargetGroups } = await client.send(
          new DescribeTargetGroupsCommand({
            Names: [NAMES.loadBalancerTargetGroupName],
          }),
        );
        await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
          client.send(
            new DeleteTargetGroupCommand({
              TargetGroupArn: TargetGroups[0].TargetGroupArn,
            }),
          );
      } catch (e) {
        state.deleteLoadBalancerTargetGroupError = e;
      }
    }),
    new ScenarioOutput("deleteLoadBalancerTargetGroupResult", (state) => {
      if (state.deleteLoadBalancerTargetGroupError) {
        console.error(state.deleteLoadBalancerTargetGroupError);
        return MESSAGES.deleteLoadBalancerTargetGroupError.replace(
          "${TARGET_GROUP_NAME}",
          NAMES.loadBalancerTargetGroupName,
        );
      }
      return MESSAGES.deletedLoadBalancerTargetGroup.replace(
        "${TARGET_GROUP_NAME}",

```

```
    NAMES.loadBalancerTargetGroupName,
  );
}),
new ScenarioAction("detachSsmOnlyRoleFromProfile", async (state) => {
  try {
    const client = new IAMClient({});
    await client.send(
      new RemoveRoleFromInstanceProfileCommand({
        InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
        RoleName: NAMES.ssmOnlyRoleName,
      }),
    );
  } catch (e) {
    state.detachSsmOnlyRoleFromProfileError = e;
  }
}),
new ScenarioOutput("detachSsmOnlyRoleFromProfileResult", (state) => {
  if (state.detachSsmOnlyRoleFromProfileError) {
    console.error(state.detachSsmOnlyRoleFromProfileError);
    return MESSAGES.detachSsmOnlyRoleFromProfileError
      .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
      .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
  }
  return MESSAGES.detachedSsmOnlyRoleFromProfile
    .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
    .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
}),
new ScenarioAction("detachSsmOnlyCustomRolePolicy", async (state) => {
  try {
    const iamClient = new IAMClient({});
    const ssmOnlyPolicy = await findPolicy(NAMES.ssmOnlyPolicyName);
    await iamClient.send(
      new DetachRolePolicyCommand({
        RoleName: NAMES.ssmOnlyRoleName,
        PolicyArn: ssmOnlyPolicy.Arn,
      }),
    );
  } catch (e) {
    state.detachSsmOnlyCustomRolePolicyError = e;
  }
}),
new ScenarioOutput("detachSsmOnlyCustomRolePolicyResult", (state) => {
  if (state.detachSsmOnlyCustomRolePolicyError) {
    console.error(state.detachSsmOnlyCustomRolePolicyError);
  }
})
```

```

    return MESSAGES.detachSsmOnlyCustomRolePolicyError
      .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
      .replace("${POLICY_NAME}", NAMES.ssmOnlyPolicyName);
  }
  return MESSAGES.detachedSsmOnlyCustomRolePolicy
    .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
    .replace("${POLICY_NAME}", NAMES.ssmOnlyPolicyName);
}),
new ScenarioAction("detachSsmOnlyAWSRolePolicy", async (state) => {
  try {
    const iamClient = new IAMClient({});
    await iamClient.send(
      new DetachRolePolicyCommand({
        RoleName: NAMES.ssmOnlyRoleName,
        PolicyArn: "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore",
      }),
    );
  } catch (e) {
    state.detachSsmOnlyAWSRolePolicyError = e;
  }
}),
new ScenarioOutput("detachSsmOnlyAWSRolePolicyResult", (state) => {
  if (state.detachSsmOnlyAWSRolePolicyError) {
    console.error(state.detachSsmOnlyAWSRolePolicyError);
    return MESSAGES.detachSsmOnlyAWSRolePolicyError
      .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
      .replace("${POLICY_NAME}", "AmazonSSMManagedInstanceCore");
  }
  return MESSAGES.detachedSsmOnlyAWSRolePolicy
    .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
    .replace("${POLICY_NAME}", "AmazonSSMManagedInstanceCore");
}),
new ScenarioAction("deleteSsmOnlyInstanceProfile", async (state) => {
  try {
    const iamClient = new IAMClient({});
    await iamClient.send(
      new DeleteInstanceProfileCommand({
        InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
      }),
    );
  } catch (e) {
    state.deleteSsmOnlyInstanceProfileError = e;
  }
}),

```

```
new ScenarioOutput("deleteSsmOnlyInstanceProfileResult", (state) => {
  if (state.deleteSsmOnlyInstanceProfileError) {
    console.error(state.deleteSsmOnlyInstanceProfileError);
    return MESSAGES.deleteSsmOnlyInstanceProfileError.replace(
      "${INSTANCE_PROFILE_NAME}",
      NAMES.ssmOnlyInstanceProfileName,
    );
  }
  return MESSAGES.deletedSsmOnlyInstanceProfile.replace(
    "${INSTANCE_PROFILE_NAME}",
    NAMES.ssmOnlyInstanceProfileName,
  );
}),
new ScenarioAction("deleteSsmOnlyPolicy", async (state) => {
  try {
    const iamClient = new IAMClient({});
    const ssmOnlyPolicy = await findPolicy(NAMES.ssmOnlyPolicyName);
    await iamClient.send(
      new DeletePolicyCommand({
        PolicyArn: ssmOnlyPolicy.Arn,
      }),
    );
  } catch (e) {
    state.deleteSsmOnlyPolicyError = e;
  }
}),
new ScenarioOutput("deleteSsmOnlyPolicyResult", (state) => {
  if (state.deleteSsmOnlyPolicyError) {
    console.error(state.deleteSsmOnlyPolicyError);
    return MESSAGES.deleteSsmOnlyPolicyError.replace(
      "${POLICY_NAME}",
      NAMES.ssmOnlyPolicyName,
    );
  }
  return MESSAGES.deletedSsmOnlyPolicy.replace(
    "${POLICY_NAME}",
    NAMES.ssmOnlyPolicyName,
  );
}),
new ScenarioAction("deleteSsmOnlyRole", async (state) => {
  try {
    const iamClient = new IAMClient({});
    await iamClient.send(
      new DeleteRoleCommand({
```



```

        RoleName: NAMES.ssmOnlyRoleName,
    )),
    );
} catch (e) {
    state.deleteSsmOnlyRoleError = e;
}
}),
new ScenarioOutput("deleteSsmOnlyRoleResult", (state) => {
    if (state.deleteSsmOnlyRoleError) {
        console.error(state.deleteSsmOnlyRoleError);
        return MESSAGES.deleteSsmOnlyRoleError.replace(
            "${ROLE_NAME}",
            NAMES.ssmOnlyRoleName,
        );
    }
    return MESSAGES.deletedSsmOnlyRole.replace(
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
    );
}),
new ScenarioAction(
    "revokeSecurityGroupIngress",
    async (
        /** @type {{ myIp: string, defaultSecurityGroup: { GroupId: string } }} */
state,
    ) => {
        const ec2Client = new EC2Client({});

        try {
            await ec2Client.send(
                new RevokeSecurityGroupIngressCommand({
                    GroupId: state.defaultSecurityGroup.GroupId,
                    CidrIp: `${state.myIp}/32`,
                    FromPort: 80,
                    ToPort: 80,
                    IpProtocol: "tcp",
                }),
            );
        } catch (e) {
            state.revokeSecurityGroupIngressError = e;
        }
    },
),
new ScenarioOutput("revokeSecurityGroupIngressResult", (state) => {

```

```

    if (state.revokeSecurityGroupIngressError) {
      console.error(state.revokeSecurityGroupIngressError);
      return MESSAGES.revokeSecurityGroupIngressError.replace(
        "${IP}",
        state.myIp,
      );
    }
    return MESSAGES.revokedSecurityGroupIngress.replace("${IP}", state.myIp);
  })),
];

/**
 * @param {string} policyName
 */
async function findPolicy(policyName) {
  const client = new IAMClient({});
  const paginatedPolicies = paginateListPolicies({ client }, {});
  for await (const page of paginatedPolicies) {
    const policy = page.Policies.find((p) => p.PolicyName === policyName);
    if (policy) {
      return policy;
    }
  }
}

/**
 * @param {string} groupName
 */
async function deleteAutoScalingGroup(groupName) {
  const client = new AutoScalingClient({});
  try {
    await client.send(
      new DeleteAutoScalingGroupCommand({
        AutoScalingGroupName: groupName,
      }),
    );
  } catch (err) {
    if (!(err instanceof Error)) {
      throw err;
    }
    console.log(err.name);
    throw err;
  }
}

```

```

/**
 * @param {string} groupName
 */
async function terminateGroupInstances(groupName) {
  const autoScalingClient = new AutoScalingClient({});
  const group = await findAutoScalingGroup(groupName);
  await autoScalingClient.send(
    new UpdateAutoScalingGroupCommand({
      AutoScalingGroupName: group.AutoScalingGroupName,
      MinSize: 0,
    }),
  );
  for (const i of group.Instances) {
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      autoScalingClient.send(
        new TerminateInstanceInAutoScalingGroupCommand({
          InstanceId: i.InstanceId,
          ShouldDecrementDesiredCapacity: true,
        }),
      ),
    );
  }
}

async function findAutoScalingGroup(groupName) {
  const client = new AutoScalingClient({});
  const paginatedGroups = paginateDescribeAutoScalingGroups({ client }, {});
  for await (const page of paginatedGroups) {
    const group = page.AutoScalingGroups.find(
      (g) => g.AutoScalingGroupName === groupName,
    );
    if (group) {
      return group;
    }
  }
  throw new Error(`Auto scaling group ${groupName} not found.`);
}

```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für JavaScript - API-Referenz.
 - [AttachLoadBalancerTargetGroups](#)

- [CreateAutoScalingGroup](#)
- [CreateInstanceProfile](#)
- [CreateLaunchTemplate](#)
- [CreateListener](#)
- [CreateLoadBalancer](#)
- [CreateTargetGroup](#)
- [DeleteAutoScalingGroup](#)
- [DeleteInstanceProfile](#)
- [DeleteLaunchTemplate](#)
- [DeleteLoadBalancer](#)
- [DeleteTargetGroup](#)
- [DescribeAutoScalingGroups](#)
- [DescribeAvailabilityZones](#)
- [DescribeIamInstanceProfileAssociations](#)
- [DescribeInstances](#)
- [DescribeLoadBalancers](#)
- [DescribeSubnets](#)
- [DescribeTargetGroups](#)
- [DescribeTargetHealth](#)
- [DescribeVpcs](#)
- [RebootInstances](#)
- [ReplaceIamInstanceProfileAssociation](#)
- [TerminateInstanceInAutoScalingGroup](#)
- [UpdateAutoScalingGroup](#)

Amazon Bedrock-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK für JavaScript (v3) mit Amazon Bedrock Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kodex finden.

Themen

- [Erste Schritte](#)
- [Aktionen](#)

Erste Schritte

Hallo Amazon Bedrock

Das folgende Codebeispiel zeigt, wie Sie mit Amazon Bedrock beginnen können.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { fileURLToPath } from "node:url";

import {
  BedrockClient,
  ListFoundationModelsCommand,
} from "@aws-sdk/client-bedrock";

const REGION = "us-east-1";
const client = new BedrockClient({ region: REGION });

export const main = async () => {
  const command = new ListFoundationModelsCommand({});
```

```

const response = await client.send(command);
const models = response.modelSummaries;

console.log("Listing the available Bedrock foundation models:");

for (const model of models) {
  console.log("=".repeat(42));
  console.log(` Model: ${model.modelId}`);
  console.log("-".repeat(42));
  console.log(` Name: ${model.modelName}`);
  console.log(` Provider: ${model.providerName}`);
  console.log(` Model ARN: ${model.modelArn}`);
  console.log(` Input modalities: ${model.inputModalities}`);
  console.log(` Output modalities: ${model.outputModalities}`);
  console.log(` Supported customizations: ${model.customizationsSupported}`);
  console.log(` Supported inference types: ${model.inferenceTypesSupported}`);
  console.log(` Lifecycle status: ${model.modelLifecycle.status}`);
  console.log(`${"=".repeat(42)}\n`);
}

const active = models.filter(
  (m) => m.modelLifecycle.status === "ACTIVE",
).length;
const legacy = models.filter(
  (m) => m.modelLifecycle.status === "LEGACY",
).length;

console.log(
  `There are ${active} active and ${legacy} legacy foundation models in ${REGION}.`,
);

return response;
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  await main();
}

```

- Einzelheiten zur API finden Sie [ListFoundationModels](#) in der AWS SDK für JavaScript API-Referenz.

Aktionen

GetFoundationModel

Das folgende Codebeispiel zeigt die Verwendung `GetFoundationModel`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Rufen Sie Details über ein Basismodell ab.

```
import { fileURLToPath } from "node:url";

import {
  BedrockClient,
  GetFoundationModelCommand,
} from "@aws-sdk/client-bedrock";

/**
 * Get details about an Amazon Bedrock foundation model.
 *
 * @return {FoundationModelDetails} - The list of available bedrock foundation
 * models.
 */
export const getFoundationModel = async () => {
  const client = new BedrockClient();

  const command = new GetFoundationModelCommand({
    modelIdentifier: "amazon.titan-embed-text-v1",
  });

  const response = await client.send(command);

  return response.modelDetails;
};
```

```
// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const model = await getFoundationModel();
  console.log(model);
}
```

- Einzelheiten zur API finden Sie [GetFoundationModel](#) in der AWS SDK für JavaScript API-Referenz.

ListFoundationModels

Das folgende Codebeispiel zeigt die Verwendung `ListFoundationModels`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Listen Sie die verfügbaren Basismodelle auf.

```
import { fileURLToPath } from "node:url";

import {
  BedrockClient,
  ListFoundationModelsCommand,
} from "@aws-sdk/client-bedrock";

/**
 * List the available Amazon Bedrock foundation models.
 *
 * @return {FoundationModelSummary[]} - The list of available bedrock foundation
  models.
 */
export const listFoundationModels = async () => {
  const client = new BedrockClient();

  const input = {
```



```
// byProvider: 'STRING_VALUE',
// byCustomizationType: 'FINE_TUNING' || 'CONTINUED_PRE_TRAINING',
// byOutputModality: 'TEXT' || 'IMAGE' || 'EMBEDDING',
// byInferenceType: 'ON_DEMAND' || 'PROVISIONED',
};

const command = new ListFoundationModelsCommand(input);

const response = await client.send(command);

return response.modelSummaries;
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const models = await listFoundationModels();
  console.log(models);
}
```

- Einzelheiten zur API finden Sie [ListFoundationModels](#) in der AWS SDK für JavaScript API-Referenz.

Amazon Bedrock Runtime-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK für JavaScript (v3) mit Amazon Bedrock Runtime Aktionen ausführen und allgemeine Szenarien implementieren.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb eines Service aufrufen oder mit anderen AWS-Services kombinieren.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kodex finden.

Themen

- [Erste Schritte](#)
- [Szenarien](#)
- [Amazon Nova](#)
- [Amazon Nova Canvas](#)
- [Anthropic Claude](#)

- [Cohere Command](#)
- [Meta Llama](#)
- [Mistral AI](#)

Erste Schritte

Hallo Amazon Bedrock

Das folgende Codebeispiel zeigt, wie Sie mit Amazon Bedrock beginnen können.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
/**
 * @typedef {Object} Content
 * @property {string} text
 *
 * @typedef {Object} Usage
 * @property {number} input_tokens
 * @property {number} output_tokens
 *
 * @typedef {Object} ResponseBody
 * @property {Content[]} content
 * @property {Usage} usage
 */

import { fileURLToPath } from "node:url";
import {
  BedrockRuntimeClient,
  InvokeModelCommand,
} from "@aws-sdk/client-bedrock-runtime";

const AWS_REGION = "us-east-1";

const MODEL_ID = "anthropic.claude-3-haiku-20240307-v1:0";
```

```
const PROMPT = "Hi. In a short paragraph, explain what you can do.";

const hello = async () => {
  console.log("=".repeat(35));
  console.log("Welcome to the Amazon Bedrock demo!");
  console.log("=".repeat(35));

  console.log("Model: Anthropic Claude 3 Haiku");
  console.log(`Prompt: ${PROMPT}\n`);
  console.log("Invoking model...\n");

  // Create a new Bedrock Runtime client instance.
  const client = new BedrockRuntimeClient({ region: AWS_REGION });

  // Prepare the payload for the model.
  const payload = {
    anthropic_version: "bedrock-2023-05-31",
    max_tokens: 1000,
    messages: [{ role: "user", content: [{ type: "text", text: PROMPT }] }],
  };

  // Invoke Claude with the payload and wait for the response.
  const apiResponse = await client.send(
    new InvokeModelCommand({
      contentType: "application/json",
      body: JSON.stringify(payload),
      modelId: MODEL_ID,
    })
  );

  // Decode and return the response(s)
  const decodedResponseBody = new TextDecoder().decode(apiResponse.body);
  /** @type {ResponseBody} */
  const responseBody = JSON.parse(decodedResponseBody);
  const responses = responseBody.content;

  if (responses.length === 1) {
    console.log(`Response: ${responses[0].text}`);
  } else {
    console.log("Haiku returned multiple responses:");
    console.log(responses);
  }

  console.log(`\nNumber of input tokens:  ${responseBody.usage.input_tokens}`);
}
```

```
    console.log(`Number of output tokens: ${responseBody.usage.output_tokens}`);  
  };  
  
  if (process.argv[1] === fileURLToPath(import.meta.url)) {  
    await hello();  
  }  
}
```

- Einzelheiten zur API finden Sie [InvokeModel](#) in der AWS SDK für JavaScript API-Referenz.

Szenarien

Aufrufen mehrerer Basismodelle in Amazon Bedrock

Das folgende Codebeispiel zeigt, wie Sie eine Aufforderung vorbereiten und an eine Vielzahl von großsprachigen Modellen (LLMs) auf Amazon Bedrock senden

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { fileURLToPath } from "node:url";  
import {  
  Scenario,  
  ScenarioAction,  
  ScenarioInput,  
  ScenarioOutput,  
} from "@aws-doc-sdk-examples/lib/scenario/index.js";  
import { FoundationModels } from "../config/foundation_models.js";  
  
/**  
 * @typedef {Object} ModelConfig  
 * @property {Function} module  
 * @property {Function} invoker  
 * @property {string} modelId  
 * @property {string} modelName
```

```

*/

const greeting = new ScenarioOutput(
  "greeting",
  "Welcome to the Amazon Bedrock Runtime client demo!",
  { header: true },
);

const selectModel = new ScenarioInput("model", "First, select a model:", {
  type: "select",
  choices: Object.values(FoundationModels).map((model) => ({
    name: model.modelName,
    value: model,
  })),
});

const enterPrompt = new ScenarioInput("prompt", "Now, enter your prompt:", {
  type: "input",
});

const printDetails = new ScenarioOutput(
  "print details",
  /**
   * @param {{ model: ModelConfig, prompt: string }} c
   */
  (c) => console.log(`Invoking ${c.model.modelName} with '${c.prompt}'...`),
);

const invokeModel = new ScenarioAction(
  "invoke model",
  /**
   * @param {{ model: ModelConfig, prompt: string, response: string }} c
   */
  async (c) => {
    const modelModule = await c.model.module();
    const invoker = c.model.invoker(modelModule);
    c.response = await invoker(c.prompt, c.model.modelId);
  },
);

const printResponse = new ScenarioOutput(
  "print response",
  /**
   * @param {{ response: string }} c

```

```
    */
    (c) => c.response,
  );

  const scenario = new Scenario("Amazon Bedrock Runtime Demo", [
    greeting,
    selectModel,
    enterPrompt,
    printDetails,
    invokeModel,
    printResponse,
  ]);

  if (process.argv[1] === fileURLToPath(import.meta.url)) {
    scenario.run();
  }
}
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für JavaScript - API-Referenz.
 - [InvokeModel](#)
 - [InvokeModelWithResponseStream](#)

Verwendung des Tools mit der Converse-API

Das folgende Codebeispiel zeigt, wie eine typische Interaktion zwischen einer Anwendung, einem generativen KI-Modell und verbundenen Tools aufgebaut oder APIs Interaktionen zwischen der KI und der Außenwelt vermittelt werden. Es wird das Beispiel der Verbindung einer externen Wetter-API mit dem KI-Modell verwendet, damit dieses auf Grundlage der Eingaben des Benutzenden Wetterinformationen in Echtzeit bereitstellen kann.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Die primäre Ausführung des Szenarioablaufs. Dieses Szenario orchestriert die Kommunikation zwischen dem Benutzer, der Converse-API von Amazon Bedrock und einem Wettertool.

```
/* Before running this JavaScript code example, set up your development environment,
including your credentials.
This demo illustrates a tool use scenario using Amazon Bedrock's Converse API and a
weather tool.
The script interacts with a foundation model on Amazon Bedrock to provide weather
information based on user
input. It uses the Open-Meteo API (https://open-meteo.com) to retrieve current
weather data for a given location.*/

import {
  Scenario,
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import {
  BedrockRuntimeClient,
  ConverseCommand,
} from "@aws-sdk/client-bedrock-runtime";

import { parseArgs } from "node:util";
import { fileURLToPath } from "node:url";
import data from "../questions.json" with { type: "json" };
import toolConfig from "../tool_config.json" with { type: "json" };

const __filename = fileURLToPath(import.meta.url);

const systemPrompt = [
  {
    text:
      "You are a weather assistant that provides current weather data for user-  

specified locations using only\n" +
      "the Weather_Tool, which expects latitude and longitude. Infer the coordinates  

from the location yourself.\n" +
      "If the user provides coordinates, infer the approximate location and refer to  

it in your response.\n" +
      "To use the tool, you strictly apply the provided tool specification.\n" +
      "If the user specifies a state, country, or region, infer the locations of  

cities within that state.\n" +
```

```

        "\n" +
        "- Explain your step-by-step process, and give brief updates before each step.
\n" +
        "- Only use the Weather_Tool for data. Never guess or make up information. \n"
        +
        "- Repeat the tool use for subsequent requests if necessary.\n" +
        "- If the tool errors, apologize, explain weather is unavailable, and suggest
other options.\n" +
        "- Report temperatures in °C (°F) and wind in km/h (mph). Keep weather reports
concise. Sparingly use\n" +
        "  emojis where appropriate.\n" +
        "- Only respond to weather queries. Remind off-topic users of your purpose.
\n" +
        "- Never claim to search online, access external data, or use tools besides
Weather_Tool.\n" +
        "- Complete the entire process until you have all required data before sending
the complete response.",
    },
];
const tools_config = toolConfig;

/// Starts the conversation with the user and handles the interaction with Bedrock.
async function askQuestion(userMessage) {
    // The maximum number of recursive calls allowed in the tool use function.
    // This helps prevent infinite loops and potential performance issues.
    const max_recursions = 5;
    const messages = [
        {
            role: "user",
            content: [{ text: userMessage }],
        },
    ];
    try {
        const response = await SendConversationtoBedrock(messages);
        await ProcessModelResponseAsync(response, messages, max_recursions);
    } catch (error) {
        console.log("error ", error);
    }
}

// Sends the conversation, the system prompt, and the tool spec to Amazon Bedrock,
// and returns the response.
// param "messages" - The conversation history including the next message to send.
// return - The response from Amazon Bedrock.

```



```

async function SendConversationtoBedrock(messages) {
  const bedRockRuntimeClient = new BedrockRuntimeClient({
    region: "us-east-1",
  });
  try {
    const modelId = "amazon.nova-lite-v1:0";
    const response = await bedRockRuntimeClient.send(
      new ConverseCommand({
        modelId: modelId,
        messages: messages,
        system: systemPrompt,
        toolConfig: tools_config,
      }),
    );
    return response;
  } catch (caught) {
    if (caught.name === "ModelNotReady") {
      console.log(
        ``${caught.name}` - Model not ready, please wait and try again.",
      );
      throw caught;
    }
    if (caught.name === "BedrockRuntimeException") {
      console.log(
        ``${caught.name}` - "Error occurred while sending Converse request.",
      );
      throw caught;
    }
  }
}

// Processes the response received via Amazon Bedrock and performs the necessary
// actions based on the stop reason.
// param "response" - The model's response returned via Amazon Bedrock.
// param "messages" - The conversation history.
// param "max_recursions" - The maximum number of recursive calls allowed.
async function ProcessModelResponseAsync(response, messages, max_recursions) {
  if (max_recursions <= 0) {
    await HandleToolUseAsync(response, messages);
  }
  if (response.stopReason === "tool_use") {
    await HandleToolUseAsync(response, messages, max_recursions - 1);
  }
  if (response.stopReason === "end_turn") {

```

```

    const messageToPrint = response.output.message.content[0].text;
    console.log(messageToPrint.replace(/<[^>]+>/g, ""));
  }
}
// Handles the tool use case by invoking the specified tool and sending the tool's
// response back to Bedrock.
// The tool response is appended to the conversation, and the conversation is sent
// back to Amazon Bedrock for further processing.
// param "response" - the model's response containing the tool use request.
// param "messages" - the conversation history.
// param "max_recursions" - The maximum number of recursive calls allowed.
async function HandleToolUseAsync(response, messages, max_recursions) {
  const toolResultFinal = [];
  try {
    const output_message = response.output.message;
    messages.push(output_message);
    const toolRequests = output_message.content;
    const toolMessage = toolRequests[0].text;
    console.log(toolMessage.replace(/<[^>]+>/g, ""));
    for (const toolRequest of toolRequests) {
      if (Object.hasOwn(toolRequest, "toolUse")) {
        const toolUse = toolRequest.toolUse;
        const latitude = toolUse.input.latitude;
        const longitude = toolUse.input.longitude;
        const toolUseID = toolUse.toolUseId;
        console.log(
          `Requesting tool ${toolUse.name}, Tool use id ${toolUseID}`,
        );
        if (toolUse.name === "Weather_Tool") {
          try {
            const current_weather = await callWeatherTool(
              longitude,
              latitude,
            ).then((current_weather) => current_weather);
            const currentWeather = current_weather;
            const toolResult = {
              toolResult: {
                toolUseId: toolUseID,
                content: [{ json: currentWeather }],
              },
            };
            toolResultFinal.push(toolResult);
          } catch (err) {
            console.log("An error occurred. ", err);
          }
        }
      }
    }
  }
}

```

```

    }
  }
}

const toolResultMessage = {
  role: "user",
  content: toolResultFinal,
};
messages.push(toolResultMessage);
// Send the conversation to Amazon Bedrock
await ProcessModelResponseAsync(
  await SendConversationtoBedrock(messages),
  messages,
);
} catch (error) {
  console.log("An error occurred. ", error);
}
}

// Call the Weathertool.
// param = longitude of location
// param = latitude of location
async function callWeatherTool(longitude, latitude) {
  // Open-Meteo API endpoint
  const apiUrl = `https://api.open-meteo.com/v1/forecast?latitude=
${latitude}&longitude=${longitude}&current_weather=true`;

  // Fetch the weather data.
  return fetch(apiUrl)
    .then((response) => {
      return response.json().then((current_weather) => {
        return current_weather;
      });
    })
    .catch((error) => {
      console.error("Error fetching weather data:", error);
    });
}

/**
 * Used repeatedly to have the user press enter.
 * @type {ScenarioInput}
 */
const pressEnter = new ScenarioInput("continue", "Press Enter to continue", {
  type: "input",

```

```
    default: "",
  });

const greet = new ScenarioOutput(
  "greet",
  "Welcome to the Amazon Bedrock Tool Use demo! \n" +
    "This assistant provides current weather information for user-specified locations. " +
    "You can ask for weather details by providing the location name or coordinates." +
    "Weather information will be provided using a custom Tool and open-meteo API." +
    "For the purposes of this example, we'll use in order the questions in ./questions.json :\n" +
    "What's the weather like in Seattle? " +
    "What's the best kind of cat? " +
    "Where is the warmest city in Washington State right now? " +
    "What's the warmest city in California right now?\n" +
    "To exit the program, simply type 'x' and press Enter.\n" +
    "Have fun and experiment with the app by editing the questions in ./questions.json! " +
    "P.S.: You're not limited to single locations, or even to using English! ",

  { header: true },
);

const displayAskQuestion1 = new ScenarioOutput(
  "displayAskQuestion1",
  "Press enter to ask question number 1 (default is 'What's the weather like in Seattle?')",
);

const askQuestion1 = new ScenarioAction(
  "askQuestion1",
  async (** @type {State} */ state) => {
    const userMessage1 = data.questions["question-1"];
    await askQuestion(userMessage1);
  },
);

const displayAskQuestion2 = new ScenarioOutput(
  "displayAskQuestion2",
  "Press enter to ask question number 2 (default is 'What's the best kind of cat?')",
);
```

```
const askQuestion2 = new ScenarioAction(
  "askQuestion2",
  async (** @type {State} */ state) => {
    const userMessage2 = data.questions["question-2"];
    await askQuestion(userMessage2);
  },
);

const displayAskQuestion3 = new ScenarioOutput(
  "displayAskQuestion3",
  "Press enter to ask question number 3 (default is 'Where is the warmest city in Washington State right now?')",
);

const askQuestion3 = new ScenarioAction(
  "askQuestion3",
  async (** @type {State} */ state) => {
    const userMessage3 = data.questions["question-3"];
    await askQuestion(userMessage3);
  },
);

const displayAskQuestion4 = new ScenarioOutput(
  "displayAskQuestion4",
  "Press enter to ask question number 4 (default is 'What's the warmest city in California right now?')",
);

const askQuestion4 = new ScenarioAction(
  "askQuestion4",
  async (** @type {State} */ state) => {
    const userMessage4 = data.questions["question-4"];
    await askQuestion(userMessage4);
  },
);

const goodbye = new ScenarioOutput(
  "goodbye",
  "Thank you for checking out the Amazon Bedrock Tool Use demo. We hope you\n" +
    "learned something new, or got some inspiration for your own apps today!\n" +
    "For more Bedrock examples in different programming languages, have a look at:\n" +
    "https://docs.aws.amazon.com/bedrock/latest/userguide/service_code_examples.html",
);
```

```
const myScenario = new Scenario("Converse Tool Scenario", [
  greet,
  pressEnter,
  displayAskQuestion1,
  askQuestion1,
  pressEnter,
  displayAskQuestion2,
  askQuestion2,
  pressEnter,
  displayAskQuestion3,
  askQuestion3,
  pressEnter,
  displayAskQuestion4,
  askQuestion4,
  pressEnter,
  goodbye,
]);

/** @type {{ stepHandlerOptions: StepHandlerOptions }} */
export const main = async (stepHandlerOptions) => {
  await myScenario.run(stepHandlerOptions);
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const { values } = parseArgs({
    options: {
      yes: {
        type: "boolean",
        short: "y",
      },
    },
  });
  main({ confirmAll: values.yes });
}
```

- Weitere API-Informationen finden Sie unter [Converse](#) in der AWS SDK für JavaScript -API-Referenz.

Amazon Nova

Converse

Das folgende Codebeispiel zeigt, wie mit der Converse-API von Bedrock eine Textnachricht an Amazon Nova gesendet wird.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Senden Sie mithilfe der Converse-API von Bedrock eine Textnachricht an Amazon Nova.

```
// This example demonstrates how to use the Amazon Nova foundation models to
// generate text.
// It shows how to:
// - Set up the Amazon Bedrock runtime client
// - Create a message
// - Configure and send a request
// - Process the response

import {
  BedrockRuntimeClient,
  ConversationRole,
  ConverseCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Step 1: Create the Amazon Bedrock runtime client
// Credentials will be automatically loaded from the environment.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Step 2: Specify which model to use:
// Available Amazon Nova models and their characteristics:
// - Amazon Nova Micro: Text-only model optimized for lowest latency and cost
// - Amazon Nova Lite: Fast, low-cost multimodal model for image, video, and text
// - Amazon Nova Pro: Advanced multimodal model balancing accuracy, speed, and
// cost
//
```

```
// For the most current model IDs, see:
// https://docs.aws.amazon.com/bedrock/latest/userguide/models-supported.html
const modelId = "amazon.nova-lite-v1:0";

// Step 3: Create the message
// The message includes the text prompt and specifies that it comes from the user
const inputText =
  "Describe the purpose of a 'hello world' program in one line.";
const message = {
  content: [{ text: inputText }],
  role: ConversationRole.USER,
};

// Step 4: Configure the request
// Optional parameters to control the model's response:
// - maxTokens: maximum number of tokens to generate
// - temperature: randomness (max: 1.0, default: 0.7)
// OR
// - topP: diversity of word choice (max: 1.0, default: 0.9)
// Note: Use either temperature OR topP, but not both
const request = {
  modelId,
  messages: [message],
  inferenceConfig: {
    maxTokens: 500, // The maximum response length
    temperature: 0.5, // Using temperature for randomness control
    //topP: 0.9,      // Alternative: use topP instead of temperature
  },
};

// Step 5: Send and process the request
// - Send the request to the model
// - Extract and return the generated text from the response
try {
  const response = await client.send(new ConverseCommand(request));
  console.log(response.output.message.content[0].text);
} catch (error) {
  console.error(`ERROR: Can't invoke '${modelId}'. Reason: ${error.message}`);
  throw error;
}
```


Senden Sie mithilfe der Converse-API von Bedrock und einer Tool-Konfiguration eine Konversation mit Nachrichten an Amazon Nova.

```
// This example demonstrates how to send a conversation of messages to Amazon Nova
// using Bedrock's Converse API with a tool configuration.
// It shows how to:
// - 1. Set up the Amazon Bedrock runtime client
// - 2. Define the parameters required enable Amazon Bedrock to use a tool when
//     formulating its response (model ID, user input, system prompt, and the tool spec)
// - 3. Send the request to Amazon Bedrock, and returns the response.
// - 4. Add the tool response to the conversation, and send it back to Amazon
//     Bedrock.
// - 5. Publish the response.

import {
  BedrockRuntimeClient,
  ConverseCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Step 1: Create the Amazon Bedrock runtime client

// Credentials will be automatically loaded from the environment
const bedRockRuntimeClient = new BedrockRuntimeClient({
  region: "us-east-1",
});

// Step 2. Define the parameters required enable Amazon Bedrock to use a tool when
// formulating its response.

// The Bedrock Model ID.
const modelId = "amazon.nova-lite-v1:0";

// The system prompt to help Amazon Bedrock craft it's response.
const system_prompt = [
  {
    text:
      "You are a music expert that provides the most popular song played on a radio" +
      "station, using only the\n" +
      "the top_song tool, which he call sign for the radio station for which you" +
      "want the most popular song. " +
      "Example calls signs are WZPZ and WKRP. \n" +
      "- Only use the top_song tool. Never guess or make up information. \n" +
```

```

    "- If the tool errors, apologize, explain weather is unavailable, and suggest
    other options.\n" +
    "- Only respond to queries about the most popular song played on a radio
    station\n" +
    "Remind off-topic users of your purpose. \n" +
    "- Never claim to search online, access external data, or use tools besides
    the top_song tool.\n",
  },
];
// The user's question.
const message = [
  {
    role: "user",
    content: [{ text: "What is the most popular song on WZPZ?" }],
  },
];
// The tool specification. In this case, it uses an example schema for
// a tool that gets the most popular song played on a radio station.
const tool_config = {
  tools: [
    {
      toolSpec: {
        name: "top_song",
        description: "Get the most popular song played on a radio station.",
        inputSchema: {
          json: {
            type: "object",
            properties: {
              sign: {
                type: "string",
                description:
                  "The call sign for the radio station for which you want the most
                  popular song. Example calls signs are WZPZ and WKRP.",
              },
            },
            required: ["sign"],
          },
        },
      },
    },
  ],
};

// Helper function to return the song and artist from top_song tool.

```

```

async function get_top_song(call_sign) {
  try {
    if (call_sign === "WZPZ") {
      const song = "Elemental Hotel";
      const artist = "8 Storey Hike";
      return { song, artist };
    }
  } catch (error) {
    console.log(`${error.message}`);
  }
}

// 3. Send the request to Amazon Bedrock, and returns the response.
export async function SendConversationtoBedrock(
  modelId,
  message,
  system_prompt,
  tool_config,
) {
  try {
    const response = await bedRockRuntimeClient.send(
      new ConverseCommand({
        modelId: modelId,
        messages: message,
        system: system_prompt,
        toolConfig: tool_config,
      }),
    );
    if (response.stopReason === "tool_use") {
      const toolResultFinal = [];
      try {
        const output_message = response.output.message;
        message.push(output_message);
        const toolRequests = output_message.content;
        const toolMessage = toolRequests[0].text;
        console.log(toolMessage.replace(/<[^>]+>/g, ""));
        for (const toolRequest of toolRequests) {
          if (Object.hasOwn(toolRequest, "toolUse")) {
            const toolUse = toolRequest.toolUse;
            const sign = toolUse.input.sign;
            const toolUseID = toolUse.toolUseId;
            console.log(
              `Requesting tool ${toolUse.name}, Tool use id ${toolUseID}`,
            );

```

```

        if (toolUse.name === "top_song") {
            const toolResult = [];
            try {
                const top_song = await get_top_song(toolUse.input.sign).then(
                    (top_song) => top_song,
                );
                const toolResult = {
                    toolResult: {
                        toolUseId: toolUseID,
                        content: [
                            {
                                json: { song: top_song.song, artist: top_song.artist },
                            },
                        ],
                    },
                };
                toolResultFinal.push(toolResult);
            } catch (err) {
                const toolResult = {
                    toolUseId: toolUseID,
                    content: [{ json: { text: err.message } }],
                    status: "error",
                };
            }
        }
    }
}

const toolResultMessage = {
    role: "user",
    content: toolResultFinal,
};

// Step 4. Add the tool response to the conversation, and send it back to
Amazon Bedrock.

message.push(toolResultMessage);
await SendConversationtoBedrock(
    modelId,
    message,
    system_prompt,
    tool_config,
);
} catch (caught) {
    console.error(`${caught.message}`);
    throw caught;
}

```

```

    }
  }

  // 4. Publish the response.
  if (response.stopReason === "end_turn") {
    const finalMessage = response.output.message.content[0].text;
    const messageToPrint = finalMessage.replace(/<[^>]+>/g);
    console.log(messageToPrint.replace(/<[^>]+>/g));
    return messageToPrint;
  }
} catch (caught) {
  if (caught.name === "ModelNotReady") {
    console.log(
      `${caught.name} - Model not ready, please wait and try again.`
    );
    throw caught;
  }
  if (caught.name === "BedrockRuntimeException") {
    console.log(
      `${caught.name} - Error occurred while sending Converse request`
    );
    throw caught;
  }
}
}
}
await SendConversationtoBedrock(modelId, message, system_prompt, tool_config);

```

- Details zur API finden Sie unter [Converse](#) in der AWS SDK für JavaScript -API-Referenz.

ConverseStream

Das folgende Codebeispiel zeigt, wie mit der Converse-API von Bedrock eine Textnachricht an Amazon Nova gesendet und der Antwortstream in Echtzeit verarbeitet wird.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Senden Sie mithilfe der Converse-API von Bedrock eine Textnachricht an Amazon Nova und verarbeiten Sie den Antwortstream in Echtzeit.

```
// This example demonstrates how to use the Amazon Nova foundation models
// to generate streaming text responses.
// It shows how to:
// - Set up the Amazon Bedrock runtime client
// - Create a message
// - Configure a streaming request
// - Process the streaming response

import {
  BedrockRuntimeClient,
  ConversationRole,
  ConverseStreamCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Step 1: Create the Amazon Bedrock runtime client
// Credentials will be automatically loaded from the environment
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Step 2: Specify which model to use
// Available Amazon Nova models and their characteristics:
// - Amazon Nova Micro: Text-only model optimized for lowest latency and cost
// - Amazon Nova Lite: Fast, low-cost multimodal model for image, video, and text
// - Amazon Nova Pro: Advanced multimodal model balancing accuracy, speed, and
//   cost
//
// For the most current model IDs, see:
// https://docs.aws.amazon.com/bedrock/latest/userguide/models-supported.html
const modelId = "amazon.nova-lite-v1:0";

// Step 3: Create the message
// The message includes the text prompt and specifies that it comes from the user
```

```

const inputText =
  "Describe the purpose of a 'hello world' program in one paragraph";
const message = {
  content: [{ text: inputText }],
  role: ConversationRole.USER,
};

// Step 4: Configure the streaming request
// Optional parameters to control the model's response:
// - maxTokens: maximum number of tokens to generate
// - temperature: randomness (max: 1.0, default: 0.7)
// OR
// - topP: diversity of word choice (max: 1.0, default: 0.9)
// Note: Use either temperature OR topP, but not both
const request = {
  modelId,
  messages: [message],
  inferenceConfig: {
    maxTokens: 500, // The maximum response length
    temperature: 0.5, // Using temperature for randomness control
    //topP: 0.9,      // Alternative: use topP instead of temperature
  },
};

// Step 5: Send and process the streaming request
// - Send the request to the model
// - Process each chunk of the streaming response
try {
  const response = await client.send(new ConverseStreamCommand(request));

  for await (const chunk of response.stream) {
    if (chunk.contentBlockDelta) {
      // Print each text chunk as it arrives
      process.stdout.write(chunk.contentBlockDelta.delta?.text || "");
    }
  }
} catch (error) {
  console.error(`ERROR: Can't invoke '${modelId}'. Reason: ${error.message}`);
  process.exitCode = 1;
}

```

- Einzelheiten zur API finden Sie [ConverseStream](#) in der AWS SDK für JavaScript API-Referenz.

Szenario: Verwendung des Tools mit der Converse-API

Das folgende Codebeispiel zeigt, wie eine typische Interaktion zwischen einer Anwendung, einem generativen KI-Modell und verbundenen Tools aufgebaut oder APIs Interaktionen zwischen der KI und der Außenwelt vermittelt werden. Es wird das Beispiel der Verbindung einer externen Wetter-API mit dem KI-Modell verwendet, damit dieses auf Grundlage der Eingaben des Benutzenden Wetterinformationen in Echtzeit bereitstellen kann.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Die primäre Ausführung des Szenarioablaufs. Dieses Szenario orchestriert die Kommunikation zwischen dem Benutzer, der Converse-API von Amazon Bedrock und einem Wettertool.

```
/* Before running this JavaScript code example, set up your development environment,
including your credentials.
This demo illustrates a tool use scenario using Amazon Bedrock's Converse API and a
weather tool.
The script interacts with a foundation model on Amazon Bedrock to provide weather
information based on user
input. It uses the Open-Meteo API (https://open-meteo.com) to retrieve current
weather data for a given location.*/

import {
  Scenario,
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import {
  BedrockRuntimeClient,
  ConverseCommand,
} from "@aws-sdk/client-bedrock-runtime";

import { parseArgs } from "node:util";
import { fileURLToPath } from "node:url";
```



```

import data from "./questions.json" with { type: "json" };
import toolConfig from "./tool_config.json" with { type: "json" };

const __filename = fileURLToPath(import.meta.url);

const systemPrompt = [
  {
    text:
      "You are a weather assistant that provides current weather data for user-  

specified locations using only\n" +
      "the Weather_Tool, which expects latitude and longitude. Infer the coordinates  

from the location yourself.\n" +
      "If the user provides coordinates, infer the approximate location and refer to  

it in your response.\n" +
      "To use the tool, you strictly apply the provided tool specification.\n" +
      "If the user specifies a state, country, or region, infer the locations of  

cities within that state.\n" +
      "\n" +
      "- Explain your step-by-step process, and give brief updates before each step.  

\n" +
      "- Only use the Weather_Tool for data. Never guess or make up information. \n"
    +
      "- Repeat the tool use for subsequent requests if necessary.\n" +
      "- If the tool errors, apologize, explain weather is unavailable, and suggest  

other options.\n" +
      "- Report temperatures in °C (°F) and wind in km/h (mph). Keep weather reports  

concise. Sparingly use\n" +
      " emojis where appropriate.\n" +
      "- Only respond to weather queries. Remind off-topic users of your purpose.  

\n" +
      "- Never claim to search online, access external data, or use tools besides  

Weather_Tool.\n" +
      "- Complete the entire process until you have all required data before sending  

the complete response.",
  },
];

const tools_config = toolConfig;

/// Starts the conversation with the user and handles the interaction with Bedrock.
async function askQuestion(userMessage) {
  // The maximum number of recursive calls allowed in the tool use function.
  // This helps prevent infinite loops and potential performance issues.
  const max_recursions = 5;
  const messages = [

```

```

    {
      role: "user",
      content: [{ text: userMessage }],
    },
  ];
  try {
    const response = await SendConversationtoBedrock(messages);
    await ProcessModelResponseAsync(response, messages, max_recurSIONs);
  } catch (error) {
    console.log("error ", error);
  }
}

// Sends the conversation, the system prompt, and the tool spec to Amazon Bedrock,
// and returns the response.
// param "messages" - The conversation history including the next message to send.
// return - The response from Amazon Bedrock.
async function SendConversationtoBedrock(messages) {
  const bedRockRuntimeClient = new BedrockRuntimeClient({
    region: "us-east-1",
  });
  try {
    const modelId = "amazon.nova-lite-v1:0";
    const response = await bedRockRuntimeClient.send(
      new ConverseCommand({
        modelId: modelId,
        messages: messages,
        system: systemPrompt,
        toolConfig: tools_config,
      }),
    );
    return response;
  } catch (caught) {
    if (caught.name === "ModelNotReady") {
      console.log(
        "`${caught.name}` - Model not ready, please wait and try again.",
      );
      throw caught;
    }
    if (caught.name === "BedrockRuntimeException") {
      console.log(
        "`${caught.name}` - Error occurred while sending Converse request.",
      );
      throw caught;
    }
  }
}

```

```

    }
  }
}

// Processes the response received via Amazon Bedrock and performs the necessary
// actions based on the stop reason.
// param "response" - The model's response returned via Amazon Bedrock.
// param "messages" - The conversation history.
// param "max_recursions" - The maximum number of recursive calls allowed.
async function ProcessModelResponseAsync(response, messages, max_recursions) {
  if (max_recursions <= 0) {
    await HandleToolUseAsync(response, messages);
  }
  if (response.stopReason === "tool_use") {
    await HandleToolUseAsync(response, messages, max_recursions - 1);
  }
  if (response.stopReason === "end_turn") {
    const messageToPrint = response.output.message.content[0].text;
    console.log(messageToPrint.replace(/<[^>]+>/g, ""));
  }
}

// Handles the tool use case by invoking the specified tool and sending the tool's
// response back to Bedrock.
// The tool response is appended to the conversation, and the conversation is sent
// back to Amazon Bedrock for further processing.
// param "response" - the model's response containing the tool use request.
// param "messages" - the conversation history.
// param "max_recursions" - The maximum number of recursive calls allowed.
async function HandleToolUseAsync(response, messages, max_recursions) {
  const toolResultFinal = [];
  try {
    const output_message = response.output.message;
    messages.push(output_message);
    const toolRequests = output_message.content;
    const toolMessage = toolRequests[0].text;
    console.log(toolMessage.replace(/<[^>]+>/g, ""));
    for (const toolRequest of toolRequests) {
      if (Object.hasOwn(toolRequest, "toolUse")) {
        const toolUse = toolRequest.toolUse;
        const latitude = toolUse.input.latitude;
        const longitude = toolUse.input.longitude;
        const toolUseID = toolUse.toolUseId;
        console.log(
          `Requesting tool ${toolUse.name}, Tool use id ${toolUseID}`,

```

```

    );
    if (toolUse.name === "Weather_Tool") {
      try {
        const current_weather = await callWeatherTool(
          longitude,
          latitude,
        ).then((current_weather) => current_weather);
        const currentWeather = current_weather;
        const toolResult = {
          toolResult: {
            toolUseId: toolUseID,
            content: [{ json: currentWeather }],
          },
        };
        toolResultFinal.push(toolResult);
      } catch (err) {
        console.log("An error occurred. ", err);
      }
    }
  }
}

const toolResultMessage = {
  role: "user",
  content: toolResultFinal,
};
messages.push(toolResultMessage);
// Send the conversation to Amazon Bedrock
await ProcessModelResponseAsync(
  await SendConversationtoBedrock(messages),
  messages,
);
} catch (error) {
  console.log("An error occurred. ", error);
}
}

// Call the Weathertool.
// param = longitude of location
// param = latitude of location
async function callWeatherTool(longitude, latitude) {
  // Open-Meteo API endpoint
  const apiUrl = `https://api.open-meteo.com/v1/forecast?latitude=${latitude}&longitude=${longitude}&current_weather=true`;

```

```

// Fetch the weather data.
return fetch(apiUrl)
  .then((response) => {
    return response.json().then((current_weather) => {
      return current_weather;
    });
  })
  .catch((error) => {
    console.error("Error fetching weather data:", error);
  });
}
/**
 * Used repeatedly to have the user press enter.
 * @type {ScenarioInput}
 */
const pressEnter = new ScenarioInput("continue", "Press Enter to continue", {
  type: "input",
  default: "",
});

const greet = new ScenarioOutput(
  "greet",
  "Welcome to the Amazon Bedrock Tool Use demo! \n" +
    "This assistant provides current weather information for user-specified locations. " +
    "You can ask for weather details by providing the location name or coordinates." +
    "Weather information will be provided using a custom Tool and open-meteo API." +
    "For the purposes of this example, we'll use in order the questions in ./questions.json :\n" +
    "What's the weather like in Seattle? " +
    "What's the best kind of cat? " +
    "Where is the warmest city in Washington State right now? " +
    "What's the warmest city in California right now?\n" +
    "To exit the program, simply type 'x' and press Enter.\n" +
    "Have fun and experiment with the app by editing the questions in ./questions.json! " +
    "P.S.: You're not limited to single locations, or even to using English! ",
  { header: true },
);
const displayAskQuestion1 = new ScenarioOutput(
  "displayAskQuestion1",

```

```
    "Press enter to ask question number 1 (default is 'What's the weather like in
    Seattle?')",
  );

const askQuestion1 = new ScenarioAction(
  "askQuestion1",
  async (** @type {State} */ state) => {
    const userMessage1 = data.questions["question-1"];
    await askQuestion(userMessage1);
  },
);

const displayAskQuestion2 = new ScenarioOutput(
  "displayAskQuestion2",
  "Press enter to ask question number 2 (default is 'What's the best kind of
  cat?')",
);

const askQuestion2 = new ScenarioAction(
  "askQuestion2",
  async (** @type {State} */ state) => {
    const userMessage2 = data.questions["question-2"];
    await askQuestion(userMessage2);
  },
);

const displayAskQuestion3 = new ScenarioOutput(
  "displayAskQuestion3",
  "Press enter to ask question number 3 (default is 'Where is the warmest city in
  Washington State right now?')",
);

const askQuestion3 = new ScenarioAction(
  "askQuestion3",
  async (** @type {State} */ state) => {
    const userMessage3 = data.questions["question-3"];
    await askQuestion(userMessage3);
  },
);

const displayAskQuestion4 = new ScenarioOutput(
  "displayAskQuestion4",
  "Press enter to ask question number 4 (default is 'What's the warmest city in
  California right now?')",
);
```

```
const askQuestion4 = new ScenarioAction(
  "askQuestion4",
  async (** @type {State} */ state) => {
    const userMessage4 = data.questions["question-4"];
    await askQuestion(userMessage4);
  },
);

const goodbye = new ScenarioOutput(
  "goodbye",
  "Thank you for checking out the Amazon Bedrock Tool Use demo. We hope you\n" +
    "learned something new, or got some inspiration for your own apps today!\n" +
    "For more Bedrock examples in different programming languages, have a look at:\n" +
    "https://docs.aws.amazon.com/bedrock/latest/userguide/service_code_examples.html",
);

const myScenario = new Scenario("Converse Tool Scenario", [
  greet,
  pressEnter,
  displayAskQuestion1,
  askQuestion1,
  pressEnter,
  displayAskQuestion2,
  askQuestion2,
  pressEnter,
  displayAskQuestion3,
  askQuestion3,
  pressEnter,
  displayAskQuestion4,
  askQuestion4,
  pressEnter,
  goodbye,
]);

/** @type {{ stepHandlerOptions: StepHandlerOptions }} */
export const main = async (stepHandlerOptions) => {
  await myScenario.run(stepHandlerOptions);
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
```

```
const { values } = parseArgs({
  options: {
    yes: {
      type: "boolean",
      short: "y",
    },
  },
});
main({ confirmAll: values.yes });
}
```

- Weitere API-Informationen finden Sie unter [Converse](#) in der AWS SDK für JavaScript -API-Referenz.

Amazon Nova Canvas

InvokeModel

Das folgende Codebeispiel zeigt, wie Amazon Nova Canvas auf Amazon Bedrock aufgerufen wird, um ein Image zu generieren.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie ein Bild mit Amazon Nova Canvas.

```
import {
  BedrockRuntimeClient,
  InvokeModelCommand,
} from "@aws-sdk/client-bedrock-runtime";
import { saveImage } from "../../utils/image-creation.js";
import { fileURLToPath } from "node:url";

/**
```



```

* This example demonstrates how to use Amazon Nova Canvas to generate images.
* It shows how to:
* - Set up the Amazon Bedrock runtime client
* - Configure the image generation parameters
* - Send a request to generate an image
* - Process the response and handle the generated image
*
* @returns {Promise<string>} Base64-encoded image data
*/
export const invokeModel = async () => {
  // Step 1: Create the Amazon Bedrock runtime client
  // Credentials will be automatically loaded from the environment
  const client = new BedrockRuntimeClient({ region: "us-east-1" });

  // Step 2: Specify which model to use
  // For the latest available models, see:
  // https://docs.aws.amazon.com/bedrock/latest/userguide/models-supported.html
  const modelId = "amazon.nova-canvas-v1:0";

  // Step 3: Configure the request payload
  // First, set the main parameters:
  // - prompt: Text description of the image to generate
  // - seed: Random number for reproducible generation (0 to 858,993,459)
  const prompt = "A stylized picture of a cute old steampunk robot";
  const seed = Math.floor(Math.random() * 858993460);

  // Then, create the payload using the following structure:
  // - taskType: TEXT_IMAGE (specifies text-to-image generation)
  // - textToImageParams: Contains the text prompt
  // - imageGenerationConfig: Contains optional generation settings (seed, quality,
  etc.)
  // For a list of available request parameters, see:
  // https://docs.aws.amazon.com/nova/latest/userguide/image-gen-req-resp-
  structure.html
  const payload = {
    taskType: "TEXT_IMAGE",
    textToImageParams: {
      text: prompt,
    },
    imageGenerationConfig: {
      seed,
      quality: "standard",
    },
  };
};

```

```

// Step 4: Send and process the request
// - Embed the payload in a request object
// - Send the request to the model
// - Extract and return the generated image data from the response
try {
  const request = {
    modelId,
    body: JSON.stringify(payload),
  };
  const response = await client.send(new InvokeModelCommand(request));

  const decodedResponseBody = new TextDecoder().decode(response.body);
  // The response includes an array of base64-encoded PNG images
  /** @type {{images: string[]}} */
  const responseBody = JSON.parse(decodedResponseBody);
  return responseBody.images[0]; // Base64-encoded image data
} catch (error) {
  console.error(`ERROR: Can't invoke '${modelId}'. Reason: ${error.message}`);
  throw error;
}
};

// If run directly, execute the example and save the generated image
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  console.log("Generating image. This may take a few seconds...");
  invokeModel()
    .then(async (imageData) => {
      const imagePath = await saveImage(imageData, "nova-canvas");
      // Example path: javascriptv3/example_code/bedrock-runtime/output/nova-canvas/
      // image-01.png
      console.log(`Image saved to: ${imagePath}`);
    })
    .catch((error) => {
      console.error("Execution failed:", error);
      process.exitCode = 1;
    });
}

```

- Einzelheiten zur API finden Sie [InvokeModel](#) in der AWS SDK für JavaScript API-Referenz.

Anthropic Claude

Converse

Das folgende Codebeispiel zeigt, wie mit der Converse-API von Bedrock eine Textnachricht an Anthropic Claude gesendet wird.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Senden Sie eine Textnachricht an Anthropic Claude mithilfe der Converse-API von Bedrock.

```
// Use the Conversation API to send a text message to Anthropic Claude.

import {
  BedrockRuntimeClient,
  ConverseCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Claude 3 Haiku.
const modelId = "anthropic.claude-3-haiku-20240307-v1:0";

// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseCommand({
```

```
    modelId,  
    messages: conversation,  
    inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },  
  });  
  
  try {  
    // Send the command to the model and wait for the response  
    const response = await client.send(command);  
  
    // Extract and print the response text.  
    const responseText = response.output.message.content[0].text;  
    console.log(responseText);  
  } catch (err) {  
    console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);  
    process.exit(1);  
  }  
}
```

- Details zur API finden Sie unter [Converse](#) in der AWS SDK für JavaScript -API-Referenz.

ConverseStream

Das folgende Codebeispiel zeigt, wie mit der Converse-API von Bedrock eine Textnachricht an Anthropic Claude gesendet und der Antwortstream in Echtzeit verarbeitet wird.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Senden Sie eine Textnachricht an Anthropic Claude mithilfe der Converse-API von Bedrock und verarbeiten Sie den Antwortstrom in Echtzeit.

```
// Use the Conversation API to send a text message to Anthropic Claude.  
  
import {  
  BedrockRuntimeClient,
```

```
    ConverseStreamCommand,
  } from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Claude 3 Haiku.
const modelId = "anthropic.claude-3-haiku-20240307-v1:0";

// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseStreamCommand({
  modelId,
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
  // Send the command to the model and wait for the response
  const response = await client.send(command);

  // Extract and print the streamed response text in real-time.
  for await (const item of response.stream) {
    if (item.contentBlockDelta) {
      process.stdout.write(item.contentBlockDelta.delta?.text);
    }
  }
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- Einzelheiten zur API finden Sie [ConverseStream](#) in der AWS SDK für JavaScript API-Referenz.

InvokeModel

Das folgende Codebeispiel zeigt, wie mit der Invoke-Model-API eine Textnachricht an Anthropic Claude gesendet wird.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Verwenden der API zum Aufrufen eines Modells zum Senden einer Textnachricht.

```
import { fileURLToPath } from "node:url";

import { FoundationModels } from "../../config/foundation_models.js";
import {
  BedrockRuntimeClient,
  InvokeModelCommand,
  InvokeModelWithResponseStreamCommand,
} from "@aws-sdk/client-bedrock-runtime";

/**
 * @typedef {Object} ResponseContent
 * @property {string} text
 *
 * @typedef {Object} MessagesResponseBody
 * @property {ResponseContent[]} content
 *
 * @typedef {Object} Delta
 * @property {string} text
 *
 * @typedef {Object} Message
 * @property {string} role
 *
 * @typedef {Object} Chunk
 * @property {string} type
 * @property {Delta} delta
 * @property {Message} message
 */
```

```

/**
 * Invokes Anthropic Claude 3 using the Messages API.
 *
 * To learn more about the Anthropic Messages API, go to:
 * https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-anthropic-
claude-messages.html
 *
 * @param {string} prompt - The input text prompt for the model to complete.
 * @param {string} [modelId] - The ID of the model to use. Defaults to
"anthropic.claude-3-haiku-20240307-v1:0".
 */
export const invokeModel = async (
  prompt,
  modelId = "anthropic.claude-3-haiku-20240307-v1:0",
) => {
  // Create a new Bedrock Runtime client instance.
  const client = new BedrockRuntimeClient({ region: "us-east-1" });

  // Prepare the payload for the model.
  const payload = {
    anthropic_version: "bedrock-2023-05-31",
    max_tokens: 1000,
    messages: [
      {
        role: "user",
        content: [{ type: "text", text: prompt }],
      },
    ],
  };

  // Invoke Claude with the payload and wait for the response.
  const command = new InvokeModelCommand({
    contentType: "application/json",
    body: JSON.stringify(payload),
    modelId,
  });
  const apiResponse = await client.send(command);

  // Decode and return the response(s)
  const decodedResponseBody = new TextDecoder().decode(apiResponse.body);
  /** @type {MessagesResponseBody} */
  const responseBody = JSON.parse(decodedResponseBody);
  return responseBody.content[0].text;

```

```
};

/**
 * Invokes Anthropic Claude 3 and processes the response stream.
 *
 * To learn more about the Anthropic Messages API, go to:
 * https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-anthropic-claude-messages.html
 *
 * @param {string} prompt - The input text prompt for the model to complete.
 * @param {string} [modelId] - The ID of the model to use. Defaults to
 * "anthropic.claude-3-haiku-20240307-v1:0".
 */
export const invokeModelWithResponseStream = async (
  prompt,
  modelId = "anthropic.claude-3-haiku-20240307-v1:0",
) => {
  // Create a new Bedrock Runtime client instance.
  const client = new BedrockRuntimeClient({ region: "us-east-1" });

  // Prepare the payload for the model.
  const payload = {
    anthropic_version: "bedrock-2023-05-31",
    max_tokens: 1000,
    messages: [
      {
        role: "user",
        content: [{ type: "text", text: prompt }],
      },
    ],
  };

  // Invoke Claude with the payload and wait for the API to respond.
  const command = new InvokeModelWithResponseStreamCommand({
    contentType: "application/json",
    body: JSON.stringify(payload),
    modelId,
  });
  const apiResponse = await client.send(command);

  let completeMessage = "";

  // Decode and process the response stream
  for await (const item of apiResponse.body) {
```



```

/** @type Chunk */
const chunk = JSON.parse(new TextDecoder().decode(item.chunk.bytes));
const chunk_type = chunk.type;

if (chunk_type === "content_block_delta") {
  const text = chunk.delta.text;
  completeMessage = completeMessage + text;
  process.stdout.write(text);
}
}

// Return the final response
return completeMessage;
};

// Invoke the function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const prompt = 'Write a paragraph starting with: "Once upon a time..."';
  const modelId = FoundationModels.CLAUDE_3_HAIKU.modelId;
  console.log(`Prompt: ${prompt}`);
  console.log(`Model ID: ${modelId}`);

  try {
    console.log("-".repeat(53));
    const response = await invokeModel(prompt, modelId);
    console.log(`\n${"-".repeat(53)}`);
    console.log("Final structured response:");
    console.log(response);
  } catch (err) {
    console.log(`\n${err}`);
  }
}

```

- Einzelheiten zur API finden Sie [InvokeModel](#) in der AWS SDK für JavaScript API-Referenz.

InvokeModelWithResponseStream

Das folgende Codebeispiel zeigt, wie mit der Invoke-Model-API eine Textnachricht an Anthropic Claude-Modelle gesendet und der Antwortstream gedruckt wird.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Verwenden Sie die Invoke-Model-API, um eine Textnachricht zu senden und den Antwortstream in Echtzeit zu verarbeiten.

```
import { fileURLToPath } from "node:url";

import { FoundationModels } from "../../config/foundation_models.js";
import {
  BedrockRuntimeClient,
  InvokeModelCommand,
  InvokeModelWithResponseStreamCommand,
} from "@aws-sdk/client-bedrock-runtime";

/**
 * @typedef {Object} ResponseContent
 * @property {string} text
 *
 * @typedef {Object} MessagesResponseBody
 * @property {ResponseContent[]} content
 *
 * @typedef {Object} Delta
 * @property {string} text
 *
 * @typedef {Object} Message
 * @property {string} role
 *
 * @typedef {Object} Chunk
 * @property {string} type
 * @property {Delta} delta
 * @property {Message} message
 */

/**
 * Invokes Anthropic Claude 3 using the Messages API.
```

```

*
* To learn more about the Anthropic Messages API, go to:
* https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-anthropic-claude-messages.html
*
* @param {string} prompt - The input text prompt for the model to complete.
* @param {string} [modelId] - The ID of the model to use. Defaults to
"anthropic.claude-3-haiku-20240307-v1:0".
*/
export const invokeModel = async (
  prompt,
  modelId = "anthropic.claude-3-haiku-20240307-v1:0",
) => {
  // Create a new Bedrock Runtime client instance.
  const client = new BedrockRuntimeClient({ region: "us-east-1" });

  // Prepare the payload for the model.
  const payload = {
    anthropic_version: "bedrock-2023-05-31",
    max_tokens: 1000,
    messages: [
      {
        role: "user",
        content: [{ type: "text", text: prompt }],
      },
    ],
  };

  // Invoke Claude with the payload and wait for the response.
  const command = new InvokeModelCommand({
    contentType: "application/json",
    body: JSON.stringify(payload),
    modelId,
  });
  const apiResponse = await client.send(command);

  // Decode and return the response(s)
  const decodedResponseBody = new TextDecoder().decode(apiResponse.body);
  /** @type {MessagesResponseBody} */
  const responseBody = JSON.parse(decodedResponseBody);
  return responseBody.content[0].text;
};

/**

```

```

* Invokes Anthropic Claude 3 and processes the response stream.
*
* To learn more about the Anthropic Messages API, go to:
* https://docs.aws.amazon.com/bedrock/latest/userguide/model-parameters-anthropic-claude-messages.html
*
* @param {string} prompt - The input text prompt for the model to complete.
* @param {string} [modelId] - The ID of the model to use. Defaults to
"anthropic.claude-3-haiku-20240307-v1:0".
*/
export const invokeModelWithResponseStream = async (
  prompt,
  modelId = "anthropic.claude-3-haiku-20240307-v1:0",
) => {
  // Create a new Bedrock Runtime client instance.
  const client = new BedrockRuntimeClient({ region: "us-east-1" });

  // Prepare the payload for the model.
  const payload = {
    anthropic_version: "bedrock-2023-05-31",
    max_tokens: 1000,
    messages: [
      {
        role: "user",
        content: [{ type: "text", text: prompt }],
      },
    ],
  };

  // Invoke Claude with the payload and wait for the API to respond.
  const command = new InvokeModelWithResponseStreamCommand({
    contentType: "application/json",
    body: JSON.stringify(payload),
    modelId,
  });
  const apiResponse = await client.send(command);

  let completeMessage = "";

  // Decode and process the response stream
  for await (const item of apiResponse.body) {
    /** @type Chunk */
    const chunk = JSON.parse(new TextDecoder().decode(item.chunk.bytes));
    const chunk_type = chunk.type;
  }

```

```
    if (chunk_type === "content_block_delta") {
      const text = chunk.delta.text;
      completeMessage = completeMessage + text;
      process.stdout.write(text);
    }
  }

  // Return the final response
  return completeMessage;
};

// Invoke the function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const prompt = 'Write a paragraph starting with: "Once upon a time..."';
  const modelId = FoundationModels.CLAUDE_3_HAIKU.modelId;
  console.log(`Prompt: ${prompt}`);
  console.log(`Model ID: ${modelId}`);

  try {
    console.log("-".repeat(53));
    const response = await invokeModel(prompt, modelId);
    console.log(`\n${"-".repeat(53)}`);
    console.log("Final structured response:");
    console.log(response);
  } catch (err) {
    console.log(`\n${err}`);
  }
}
```

- Einzelheiten zur API finden Sie [InvokeModelWithResponseStream](#) in der AWS SDK für JavaScript API-Referenz.

Cohere Command

Converse

Das folgende Codebeispiel zeigt, wie mit der Converse-API von Bedrock eine Textnachricht an Cohere Command gesendet wird.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Senden Sie mithilfe der Converse-API von Bedrock eine Textnachricht an Cohere Command.

```
// Use the Conversation API to send a text message to Cohere Command.

import {
  BedrockRuntimeClient,
  ConverseCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Command R.
const modelId = "cohere.command-r-v1:0";

// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseCommand({
  modelId,
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
  // Send the command to the model and wait for the response
  const response = await client.send(command);
}
```

```
// Extract and print the response text.
const responseText = response.output.message.content[0].text;
console.log(responseText);
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- Weitere API-Informationen finden Sie unter [Converse](#) in der AWS SDK für JavaScript -API-Referenz.

ConverseStream

Das folgende Codebeispiel zeigt, wie mit der Converse-API von Bedrock eine Textnachricht an Cohere Command gesendet und der Antwortstream in Echtzeit verarbeitet wird.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Senden Sie mithilfe der Converse-API von Bedrock eine Textnachricht an Cohere Command und verarbeiten Sie den Antwortstream in Echtzeit.

```
// Use the Conversation API to send a text message to Cohere Command.

import {
  BedrockRuntimeClient,
  ConverseStreamCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Command R.
```

```
const modelId = "cohere.command-r-v1:0";

// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseStreamCommand({
  modelId,
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
  // Send the command to the model and wait for the response
  const response = await client.send(command);

  // Extract and print the streamed response text in real-time.
  for await (const item of response.stream) {
    if (item.contentBlockDelta) {
      process.stdout.write(item.contentBlockDelta.delta?.text);
    }
  }
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- Einzelheiten zur API finden Sie [ConverseStream](#) in der AWS SDK für JavaScript API-Referenz.

Meta Llama

Converse

Das folgende Codebeispiel zeigt, wie mit der Converse-API von Bedrock eine Textnachricht an Meta Llama gesendet wird.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Senden Sie mithilfe der Converse-API von Bedrock eine Textnachricht an Meta Llama.

```
// Use the Conversation API to send a text message to Meta Llama.

import {
  BedrockRuntimeClient,
  ConverseCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Llama 3 8b Instruct.
const modelId = "meta.llama3-8b-instruct-v1:0";

// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseCommand({
```

```
    modelId,  
    messages: conversation,  
    inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },  
  });  
  
  try {  
    // Send the command to the model and wait for the response  
    const response = await client.send(command);  
  
    // Extract and print the response text.  
    const responseText = response.output.message.content[0].text;  
    console.log(responseText);  
  } catch (err) {  
    console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);  
    process.exit(1);  
  }  
}
```

- Details zur API finden Sie unter [Converse](#) in der AWS SDK für JavaScript -API-Referenz.

ConverseStream

Das folgende Codebeispiel zeigt, wie mit der Converse-API von Bedrock eine Textnachricht an Meta Llama gesendet und der Antwortstream in Echtzeit verarbeitet wird.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Senden Sie mithilfe der Converse-API von Bedrock eine Textnachricht an Meta Llama und verarbeiten Sie den Antwortstream in Echtzeit.

```
// Use the Conversation API to send a text message to Meta Llama.  
  
import {  
  BedrockRuntimeClient,
```

```
    ConverseStreamCommand,
  } from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Llama 3 8b Instruct.
const modelId = "meta.llama3-8b-instruct-v1:0";

// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseStreamCommand({
  modelId,
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
  // Send the command to the model and wait for the response
  const response = await client.send(command);

  // Extract and print the streamed response text in real-time.
  for await (const item of response.stream) {
    if (item.contentBlockDelta) {
      process.stdout.write(item.contentBlockDelta.delta?.text);
    }
  }
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- Einzelheiten zur API finden Sie [ConverseStream](#) in der AWS SDK für JavaScript API-Referenz.

InvokeModel

Das folgende Codebeispiel zeigt, wie mit der Invoke-Model-API eine Textnachricht an Meta Llama gesendet wird.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Verwenden der API zum Aufrufen eines Modells zum Senden einer Textnachricht.

```
// Send a prompt to Meta Llama 3 and print the response.

import {
  BedrockRuntimeClient,
  InvokeModelCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region of your choice.
const client = new BedrockRuntimeClient({ region: "us-west-2" });

// Set the model ID, e.g., Llama 3 70B Instruct.
const modelId = "meta.llama3-70b-instruct-v1:0";

// Define the user message to send.
const userMessage =
  "Describe the purpose of a 'hello world' program in one sentence.";

// Embed the message in Llama 3's prompt format.
const prompt = `
<|begin_of_text|><|start_header_id|>user<|end_header_id|>
${userMessage}
<|eot_id|>
<|start_header_id|>assistant<|end_header_id|>
`;

// Format the request payload using the model's native structure.
const request = {
  prompt,
```

```
// Optional inference parameters:
max_gen_len: 512,
temperature: 0.5,
top_p: 0.9,
};

// Encode and send the request.
const response = await client.send(
  new InvokeModelCommand({
    contentType: "application/json",
    body: JSON.stringify(request),
    modelId,
  }),
);

// Decode the native response body.
/** @type {{ generation: string }} */
const nativeResponse = JSON.parse(new TextDecoder().decode(response.body));

// Extract and print the generated text.
const responseText = nativeResponse.generation;
console.log(responseText);

// Learn more about the Llama 3 prompt format at:
// https://llama.meta.com/docs/model-cards-and-prompt-formats/meta-llama-3/#special-tokens-used-with-meta-llama-3
```

- Einzelheiten zur API finden Sie [InvokeModel](#) in der AWS SDK für JavaScript API-Referenz.

InvokeModelWithResponseStream

Das folgende Codebeispiel zeigt, wie mit der Invoke-Model-API eine Textnachricht an Meta Llama gesendet und der Antwortstream gedruckt wird.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Verwenden Sie die Invoke-Model-API, um eine Textnachricht zu senden und den Antwortstream in Echtzeit zu verarbeiten.

```
// Send a prompt to Meta Llama 3 and print the response stream in real-time.

import {
  BedrockRuntimeClient,
  InvokeModelWithResponseStreamCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region of your choice.
const client = new BedrockRuntimeClient({ region: "us-west-2" });

// Set the model ID, e.g., Llama 3 70B Instruct.
const modelId = "meta.llama3-70b-instruct-v1:0";

// Define the user message to send.
const userMessage =
  "Describe the purpose of a 'hello world' program in one sentence.";

// Embed the message in Llama 3's prompt format.
const prompt = `
<|begin_of_text|><|start_header_id|>user<|end_header_id|>
${userMessage}
<|eot_id|>
<|start_header_id|>assistant<|end_header_id|>
`;

// Format the request payload using the model's native structure.
const request = {
  prompt,
  // Optional inference parameters:
  max_gen_len: 512,
  temperature: 0.5,
  top_p: 0.9,
};

// Encode and send the request.
const responseStream = await client.send(
  new InvokeModelWithResponseStreamCommand({
    contentType: "application/json",
    body: JSON.stringify(request),
    modelId,
  })
);
```

```
    }},  
  );  
  
  // Extract and print the response stream in real-time.  
  for await (const event of responseStream.body) {  
    /** @type {{ generation: string }} */  
    const chunk = JSON.parse(new TextDecoder().decode(event.chunk.bytes));  
    if (chunk.generation) {  
      process.stdout.write(chunk.generation);  
    }  
  }  
}  
  
// Learn more about the Llama 3 prompt format at:  
// https://llama.meta.com/docs/model-cards-and-prompt-formats/meta-llama-3/#special-tokens-used-with-meta-llama-3
```

- Einzelheiten zur API finden Sie [InvokeModelWithResponseStream](#) in der AWS SDK für JavaScript API-Referenz.

Mistral AI

Converse

Das folgende Codebeispiel zeigt, wie mit der Converse-API von Bedrock eine Textnachricht an Mistral gesendet wird.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Senden Sie mithilfe der Converse-API von Bedrock eine Textnachricht an Mistral.

```
// Use the Conversation API to send a text message to Mistral.  
  
import {  
  BedrockRuntimeClient,
```

```
    ConverseCommand,
  } from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Mistral Large.
const modelId = "mistral.mistral-large-2402-v1:0";

// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseCommand({
  modelId,
  messages: conversation,
  inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
});

try {
  // Send the command to the model and wait for the response
  const response = await client.send(command);

  // Extract and print the response text.
  const responseText = response.output.message.content[0].text;
  console.log(responseText);
} catch (err) {
  console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
  process.exit(1);
}
```

- Weitere API-Informationen finden Sie unter [Converse](#) in der AWS SDK für JavaScript -API-Referenz.

ConverseStream

Das folgende Codebeispiel zeigt, wie mit der Converse-API von Bedrock eine Textnachricht an Mistral gesendet und der Antwortstream in Echtzeit verarbeitet wird.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Senden Sie mithilfe der Converse-API von Bedrock eine Textnachricht an Mistral und verarbeiten Sie den Antwortstream in Echtzeit.

```
// Use the Conversation API to send a text message to Mistral.

import {
  BedrockRuntimeClient,
  ConverseStreamCommand,
} from "@aws-sdk/client-bedrock-runtime";

// Create a Bedrock Runtime client in the AWS Region you want to use.
const client = new BedrockRuntimeClient({ region: "us-east-1" });

// Set the model ID, e.g., Mistral Large.
const modelId = "mistral.mistral-large-2402-v1:0";

// Start a conversation with the user message.
const userMessage =
  "Describe the purpose of a 'hello world' program in one line.";
const conversation = [
  {
    role: "user",
    content: [{ text: userMessage }],
  },
];

// Create a command with the model ID, the message, and a basic configuration.
const command = new ConverseStreamCommand({
  modelId,
```

```
    messages: conversation,
    inferenceConfig: { maxTokens: 512, temperature: 0.5, topP: 0.9 },
  });

  try {
    // Send the command to the model and wait for the response
    const response = await client.send(command);

    // Extract and print the streamed response text in real-time.
    for await (const item of response.stream) {
      if (item.contentBlockDelta) {
        process.stdout.write(item.contentBlockDelta.delta?.text);
      }
    }
  } catch (err) {
    console.log(`ERROR: Can't invoke '${modelId}'. Reason: ${err}`);
    process.exit(1);
  }
}
```

- Einzelheiten zur API finden Sie [ConverseStream](#) in der AWS SDK für JavaScript API-Referenz.

InvokeModel

Das folgende Codebeispiel zeigt, wie mit der Invoke-Model-API eine Textnachricht an Mistral-Modelle gesendet wird.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Verwenden der API zum Aufrufen eines Modells zum Senden einer Textnachricht.

```
import { fileURLToPath } from "node:url";

import { FoundationModels } from "../../config/foundation_models.js";
```

```
import {
  BedrockRuntimeClient,
  InvokeModelCommand,
} from "@aws-sdk/client-bedrock-runtime";

/**
 * @typedef {Object} Output
 * @property {string} text
 *
 * @typedef {Object} ResponseBody
 * @property {Output[]} outputs
 */

/**
 * Invokes a Mistral 7B Instruct model.
 *
 * @param {string} prompt - The input text prompt for the model to complete.
 * @param {string} [modelId] - The ID of the model to use. Defaults to
 * "mistral.mistral-7b-instruct-v0:2".
 */
export const invokeModel = async (
  prompt,
  modelId = "mistral.mistral-7b-instruct-v0:2",
) => {
  // Create a new Bedrock Runtime client instance.
  const client = new BedrockRuntimeClient({ region: "us-east-1" });

  // Mistral instruct models provide optimal results when embedding
  // the prompt into the following template:
  const instruction = `[INST] ${prompt} [/INST]`;

  // Prepare the payload.
  const payload = {
    prompt: instruction,
    max_tokens: 500,
    temperature: 0.5,
  };

  // Invoke the model with the payload and wait for the response.
  const command = new InvokeModelCommand({
    contentType: "application/json",
    body: JSON.stringify(payload),
    modelId,
  });
```

```
const apiResponse = await client.send(command);

// Decode and return the response.
const decodedResponseBody = new TextDecoder().decode(apiResponse.body);
/** @type {ResponseBody} */
const responseBody = JSON.parse(decodedResponseBody);
return responseBody.outputs[0].text;
};

// Invoke the function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const prompt =
    'Complete the following in one sentence: "Once upon a time...";
  const modelId = FoundationModels.MISTRAL_7B.modelId;
  console.log(`Prompt: ${prompt}`);
  console.log(`Model ID: ${modelId}`);

  try {
    console.log("-".repeat(53));
    const response = await invokeModel(prompt, modelId);
    console.log(response);
  } catch (err) {
    console.log(err);
  }
}
```

- Einzelheiten zur API finden Sie [InvokeModel](#) in der AWS SDK für JavaScript API-Referenz.

Beispiele für Amazon Bedrock Agents mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie AWS SDK für JavaScript (v3) mit Amazon Bedrock Agents verwenden.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kodex finden.

Themen

- [Erste Schritte](#)
- [Aktionen](#)

Erste Schritte

Hallo Agenten für Amazon Bedrock

Die folgenden Codebeispiele veranschaulichen die ersten Schritte mit Agenten für Amazon Bedrock.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { fileURLToPath } from "node:url";

import {
  BedrockAgentClient,
  GetAgentCommand,
  paginateListAgents,
} from "@aws-sdk/client-bedrock-agent";

/**
 * @typedef {Object} AgentSummary
 */

/**
 * A simple scenario to demonstrate basic setup and interaction with the Bedrock
 * Agents Client.
 *
 * This function first initializes the Amazon Bedrock Agents client for a specific
 * region.
 * It then retrieves a list of existing agents using the streamlined paginator
 * approach.
 * For each agent found, it retrieves detailed information using a command object.
 *
 * Demonstrates:
```

```
* - Use of the Bedrock Agents client to initialize and communicate with the AWS
service.
* - Listing resources in a paginated response pattern.
* - Accessing an individual resource using a command object.
*
* @returns {Promise<void>} A promise that resolves when the function has completed
execution.
*/
export const main = async () => {
  const region = "us-east-1";

  console.log("=".repeat(68));

  console.log(`Initializing Amazon Bedrock Agents client for ${region}...`);
  const client = new BedrockAgentClient({ region });

  console.log("Retrieving the list of existing agents...");
  const paginatorConfig = { client };
  const pages = paginateListAgents(paginatorConfig, {});

  /** @type {AgentSummary[]} */
  const agentSummaries = [];
  for await (const page of pages) {
    agentSummaries.push(...page.agentSummaries);
  }

  console.log(`Found ${agentSummaries.length} agents in ${region}.`);

  if (agentSummaries.length > 0) {
    for (const agentSummary of agentSummaries) {
      const agentId = agentSummary.agentId;
      console.log("=".repeat(68));
      console.log(`Retrieving agent with ID: ${agentId}:`);
      console.log("-".repeat(68));

      const command = new GetAgentCommand({ agentId });
      const response = await client.send(command);
      const agent = response.agent;

      console.log(` Name: ${agent.agentName}`);
      console.log(` Status: ${agent.agentStatus}`);
      console.log(` ARN: ${agent.agentArn}`);
      console.log(` Foundation model: ${agent.foundationModel}`);
    }
  }
}
```

```
    }  
    console.log("=".repeat(68));  
};  
  
// Invoke main function if this file was run directly.  
if (process.argv[1] === fileURLToPath(import.meta.url)) {  
    await main();  
}
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für JavaScript - API-Referenz.
 - [GetAgent](#)
 - [ListAgents](#)

Aktionen

CreateAgent

Das folgende Codebeispiel zeigt, wie man es benutzt `CreateAgent`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie einen Agenten.

```
import { fileURLToPath } from "node:url";  
import { checkForPlaceholders } from "../lib/utils.js";  
  
import {  
    BedrockAgentClient,  
    CreateAgentCommand,  
} from "@aws-sdk/client-bedrock-agent";  
  
/**
```

```

* Creates an Amazon Bedrock Agent.
*
* @param {string} agentName - A name for the agent that you create.
* @param {string} foundationModel - The foundation model to be used by the agent
you create.
* @param {string} agentResourceRoleArn - The ARN of the IAM role with permissions
required by the agent.
* @param {string} [region='us-east-1'] - The AWS region in use.
* @returns {Promise<import("@aws-sdk/client-bedrock-agent").Agent>} An object
containing details of the created agent.
*/
export const createAgent = async (
  agentName,
  foundationModel,
  agentResourceRoleArn,
  region = "us-east-1",
) => {
  const client = new BedrockAgentClient({ region });

  const command = new CreateAgentCommand({
    agentName,
    foundationModel,
    agentResourceRoleArn,
  });
  const response = await client.send(command);

  return response.agent;
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  // Replace the placeholders for agentName and accountId, and roleName with a
  unique name for the new agent,
  // the id of your AWS account, and the name of an existing execution role that the
  agent can use inside your account.
  // For foundationModel, specify the desired model. Ensure to remove the brackets
  '[]' before adding your data.

  // A string (max 100 chars) that can include letters, numbers, dashes '-', and
  underscores '_'.
  const agentName = "[your-bedrock-agent-name]";

  // Your AWS account id.
  const accountId = "[123456789012]";

```



```
// The name of the agent's execution role. It must be prefixed by
`AmazonBedrockExecutionRoleForAgents_`.
const roleName = "[AmazonBedrockExecutionRoleForAgents_your-role-name]";

// The ARN for the agent's execution role.
// Follow the ARN format: 'arn:aws:iam::account-id:role/role-name'
const roleArn = `arn:aws:iam::${accountId}:role/${roleName}`;

// Specify the model for the agent. Change if a different model is preferred.
const foundationModel = "anthropic.claude-v2";

// Check for unresolved placeholders in agentName and roleArn.
checkForPlaceholders([agentName, roleArn]);

console.log("Creating a new agent...");

const agent = await createAgent(agentName, foundationModel, roleArn);
console.log(agent);
}
```

- Einzelheiten zur API finden Sie [CreateAgent](#) in der AWS SDK für JavaScript API-Referenz.

DeleteAgent

Das folgende Codebeispiel zeigt die Verwendung `DeleteAgent`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Löschen eines Agenten.

```
import { fileURLToPath } from "node:url";
import { checkForPlaceholders } from "../lib/utils.js";
```

```

import {
  BedrockAgentClient,
  DeleteAgentCommand,
} from "@aws-sdk/client-bedrock-agent";

/**
 * Deletes an Amazon Bedrock Agent.
 *
 * @param {string} agentId - The unique identifier of the agent to delete.
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<import("@aws-sdk/client-bedrock-agent").DeleteAgentCommandOutput>} An object containing the agent id, the status,
  and some additional metadata.
 */
export const deleteAgent = (agentId, region = "us-east-1") => {
  const client = new BedrockAgentClient({ region });
  const command = new DeleteAgentCommand({ agentId });
  return client.send(command);
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  // Replace the placeholders for agentId with an existing agent's id.
  // Ensure to remove the brackets (`[]`) before adding your data.

  // The agentId must be an alphanumeric string with exactly 10 characters.
  const agentId = "[ABC123DE45]";

  // Check for unresolved placeholders in agentId.
  checkForPlaceholders([agentId]);

  console.log(`Deleting agent with ID ${agentId}...`);

  const response = await deleteAgent(agentId);
  console.log(response);
}

```

- Einzelheiten zur API finden Sie [DeleteAgent](#) in der AWS SDK für JavaScript API-Referenz.

GetAgent

Das folgende Codebeispiel zeigt die Verwendung `GetAgent`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Abrufen eines Agenten.

```
import { fileURLToPath } from "node:url";
import { checkForPlaceholders } from "../lib/utils.js";

import {
  BedrockAgentClient,
  GetAgentCommand,
} from "@aws-sdk/client-bedrock-agent";

/**
 * Retrieves the details of an Amazon Bedrock Agent.
 *
 * @param {string} agentId - The unique identifier of the agent.
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<import("@aws-sdk/client-bedrock-agent").Agent>} An object
  containing the agent details.
 */
export const getAgent = async (agentId, region = "us-east-1") => {
  const client = new BedrockAgentClient({ region });

  const command = new GetAgentCommand({ agentId });
  const response = await client.send(command);
  return response.agent;
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  // Replace the placeholders for agentId with an existing agent's id.
  // Ensure to remove the brackets '[]' before adding your data.

  // The agentId must be an alphanumeric string with exactly 10 characters.
  const agentId = "[ABC123DE45]";
```

```
// Check for unresolved placeholders in agentId.
checkForPlaceholders([agentId]);

console.log(`Retrieving agent with ID ${agentId}...`);

const agent = await getAgent(agentId);
console.log(agent);
}
```

- Einzelheiten zur API finden Sie [GetAgent](#) in der AWS SDK für JavaScript API-Referenz.

ListAgentActionGroups

Das folgende Codebeispiel zeigt die Verwendung `ListAgentActionGroups`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Auflisten der Aktionsgruppen für einen Agenten.

```
import { fileURLToPath } from "node:url";
import { checkForPlaceholders } from "../lib/utils.js";

import {
  BedrockAgentClient,
  ListAgentActionGroupsCommand,
  paginateListAgentActionGroups,
} from "@aws-sdk/client-bedrock-agent";

/**
 * Retrieves a list of Action Groups of an agent utilizing the paginator function.
 *
 * This function leverages a paginator, which abstracts the complexity of
 * pagination, providing
 * a straightforward way to handle paginated results inside a `for await...of` loop.
 */
```

```

*
* @param {string} agentId - The unique identifier of the agent.
* @param {string} agentVersion - The version of the agent.
* @param {string} [region='us-east-1'] - The AWS region in use.
* @returns {Promise<ActionGroupSummary[]>} An array of action group summaries.
*/
export const listAgentActionGroupsWithPaginator = async (
  agentId,
  agentVersion,
  region = "us-east-1",
) => {
  const client = new BedrockAgentClient({ region });

  // Create a paginator configuration
  const paginatorConfig = {
    client,
    pageSize: 10, // optional, added for demonstration purposes
  };

  const params = { agentId, agentVersion };

  const pages = paginateListAgentActionGroups(paginatorConfig, params);

  // Paginate until there are no more results
  const actionGroupSummaries = [];
  for await (const page of pages) {
    actionGroupSummaries.push(...page.actionGroupSummaries);
  }

  return actionGroupSummaries;
};

/**
 * Retrieves a list of Action Groups of an agent utilizing the
 * ListAgentActionGroupsCommand.
 *
 * This function demonstrates the manual approach, sending a command to the client
 * and processing the response.
 * Pagination must manually be managed. For a simplified approach that abstracts
 * away pagination logic, see
 * the `listAgentActionGroupsWithPaginator()` example below.
 *
 * @param {string} agentId - The unique identifier of the agent.
 * @param {string} agentVersion - The version of the agent.

```

```

* @param {string} [region='us-east-1'] - The AWS region in use.
* @returns {Promise<ActionGroupSummary[]>} An array of action group summaries.
*/
export const listAgentActionGroupsWithCommandObject = async (
  agentId,
  agentVersion,
  region = "us-east-1",
) => {
  const client = new BedrockAgentClient({ region });

  let nextToken;
  const actionGroupSummaries = [];
  do {
    const command = new ListAgentActionGroupsCommand({
      agentId,
      agentVersion,
      nextToken,
      maxResults: 10, // optional, added for demonstration purposes
    });

    /** @type {{actionGroupSummaries: ActionGroupSummary[], nextToken?: string}} */
    const response = await client.send(command);

    for (const actionGroup of response.actionGroupSummaries || []) {
      actionGroupSummaries.push(actionGroup);
    }

    nextToken = response.nextToken;
  } while (nextToken);

  return actionGroupSummaries;
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  // Replace the placeholders for agentId and agentVersion with an existing agent's
  id and version.
  // Ensure to remove the brackets '[]' before adding your data.

  // The agentId must be an alphanumeric string with exactly 10 characters.
  const agentId = "[ABC123DE45]";

  // A string either containing `DRAFT` or a number with 1-5 digits (e.g., '123' or
  'DRAFT').

```

```
const agentVersion = "[DRAFT]";

// Check for unresolved placeholders in agentId and agentVersion.
checkForPlaceholders([agentId, agentVersion]);

console.log("=".repeat(68));
console.log(
  "Listing agent action groups using ListAgentActionGroupsCommand:",
);

for (const actionGroup of await listAgentActionGroupsWithCommandObject(
  agentId,
  agentVersion,
)) {
  console.log(actionGroup);
}

console.log("=".repeat(68));
console.log(
  "Listing agent action groups using the paginateListAgents function:",
);
for (const actionGroup of await listAgentActionGroupsWithPaginator(
  agentId,
  agentVersion,
)) {
  console.log(actionGroup);
}
}
```

- Einzelheiten zur API finden Sie [ListAgentActionGroups](#) in der AWS SDK für JavaScript API-Referenz.

ListAgents

Das folgende Codebeispiel zeigt die Verwendung `ListAgents`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Auflisten der Agenten, die zu einem Konto gehören.

```
import { fileURLToPath } from "node:url";

import {
  BedrockAgentClient,
  ListAgentsCommand,
  paginateListAgents,
} from "@aws-sdk/client-bedrock-agent";

/**
 * Retrieves a list of available Amazon Bedrock agents utilizing the paginator
 * function.
 *
 * This function leverages a paginator, which abstracts the complexity of
 * pagination, providing
 * a straightforward way to handle paginated results inside a `for await...of` loop.
 *
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<AgentSummary[]>} An array of agent summaries.
 */
export const listAgentsWithPaginator = async (region = "us-east-1") => {
  const client = new BedrockAgentClient({ region });

  const paginatorConfig = {
    client,
    pageSize: 10, // optional, added for demonstration purposes
  };

  const pages = paginateListAgents(paginatorConfig, {});

  // Paginate until there are no more results
  const agentSummaries = [];
  for await (const page of pages) {
```



```

    agentSummaries.push(...page.agentSummaries);
  }

  return agentSummaries;
};

/**
 * Retrieves a list of available Amazon Bedrock agents utilizing the
 * ListAgentsCommand.
 *
 * This function demonstrates the manual approach, sending a command to the client
 * and processing the response.
 *
 * Pagination must manually be managed. For a simplified approach that abstracts
 * away pagination logic, see
 * the `listAgentsWithPaginator()` example below.
 *
 * @param {string} [region='us-east-1'] - The AWS region in use.
 * @returns {Promise<AgentSummary[]>} An array of agent summaries.
 */
export const listAgentsWithCommandObject = async (region = "us-east-1") => {
  const client = new BedrockAgentClient({ region });

  let nextToken;
  const agentSummaries = [];
  do {
    const command = new ListAgentsCommand({
      nextToken,
      maxResults: 10, // optional, added for demonstration purposes
    });

    /** @type {{agentSummaries: AgentSummary[], nextToken?: string}} */
    const paginatedResponse = await client.send(command);

    agentSummaries.push(...(paginatedResponse.agentSummaries || []));

    nextToken = paginatedResponse.nextToken;
  } while (nextToken);

  return agentSummaries;
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  console.log("=".repeat(68));
}

```

```
console.log("Listing agents using ListAgentsCommand:");
for (const agent of await listAgentsWithCommandObject()) {
  console.log(agent);
}

console.log("=".repeat(68));
console.log("Listing agents using the paginateListAgents function:");
for (const agent of await listAgentsWithPaginator()) {
  console.log(agent);
}
}
```

- Einzelheiten zur API finden Sie [ListAgents](#) in der AWS SDK für JavaScript API-Referenz.

Amazon Bedrock Agents Runtime-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie AWS SDK für JavaScript (v3) mit Amazon Bedrock Agents Runtime verwenden.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kodex finden.

Themen

- [Aktionen](#)

Aktionen

InvokeAgent

Das folgende Codebeispiel zeigt, wie Sie `InvokeAgent`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import {
  BedrockAgentRuntimeClient,
  InvokeAgentCommand,
} from "@aws-sdk/client-bedrock-agent-runtime";

/**
 * @typedef {Object} ResponseBody
 * @property {string} completion
 */

/**
 * Invokes a Bedrock agent to run an inference using the input
 * provided in the request body.
 *
 * @param {string} prompt - The prompt that you want the Agent to complete.
 * @param {string} sessionId - An arbitrary identifier for the session.
 */
export const invokeBedrockAgent = async (prompt, sessionId) => {
  const client = new BedrockAgentRuntimeClient({ region: "us-east-1" });
  // const client = new BedrockAgentRuntimeClient({
  //   region: "us-east-1",
  //   credentials: {
  //     accessKeyId: "accessKeyId", // permission to invoke agent
  //     secretAccessKey: "accessKeySecret",
  //   },
  // });

  const agentId = "AJBHXXILZN";
  const agentAliasId = "AVKP1ITZAA";

  const command = new InvokeAgentCommand({
    agentId,
    agentAliasId,
```

```
    sessionId,
    inputText: prompt,
  });

  try {
    let completion = "";
    const response = await client.send(command);

    if (response.completion === undefined) {
      throw new Error("Completion is undefined");
    }

    for await (const chunkEvent of response.completion) {
      const chunk = chunkEvent.chunk;
      console.log(chunk);
      const decodedResponse = new TextDecoder("utf-8").decode(chunk.bytes);
      completion += decodedResponse;
    }

    return { sessionId: sessionId, completion };
  } catch (err) {
    console.error(err);
  }
};

// Call function if run directly
import { fileURLToPath } from "node:url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const result = await invokeBedrockAgent("I need help.", "123");
  console.log(result);
}
```

- Einzelheiten zur API finden Sie [InvokeAgent](#) in der AWS SDK für JavaScript API-Referenz.

InvokeFlow

Das folgende Codebeispiel zeigt die Verwendung `InvokeFlow`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { fileURLToPath } from "node:url";

import {
  BedrockAgentRuntimeClient,
  InvokeFlowCommand,
} from "@aws-sdk/client-bedrock-agent-runtime";

/**
 * Invokes an alias of a flow to run the inputs that you specify and return
 * the output of each node as a stream.
 *
 * @param {{
 *   flowIdentifier: string,
 *   flowAliasIdentifier: string,
 *   prompt?: string,
 *   region?: string
 * }} options
 * @returns {Promise<import("@aws-sdk/client-bedrock-agent").FlowNodeOutput>} An
 * object containing information about the output from flow invocation.
 */
export const invokeBedrockFlow = async ({
  flowIdentifier,
  flowAliasIdentifier,
  prompt = "Hi, how are you?",
  region = "us-east-1",
}) => {
  const client = new BedrockAgentRuntimeClient({ region });

  const command = new InvokeFlowCommand({
    flowIdentifier,
    flowAliasIdentifier,
    inputs: [
      {

```

```

        content: {
            document: prompt,
        },
        nodeName: "FlowInputNode",
        nodeOutputName: "document",
    },
],
});

let flowResponse = {};
const response = await client.send(command);

for await (const chunkEvent of response.responseStream) {
    const { flowOutputEvent, flowCompletionEvent } = chunkEvent;

    if (flowOutputEvent) {
        flowResponse = { ...flowResponse, ...flowOutputEvent };
        console.log("Flow output event:", flowOutputEvent);
    } else if (flowCompletionEvent) {
        flowResponse = { ...flowResponse, ...flowCompletionEvent };
        console.log("Flow completion event:", flowCompletionEvent);
    }
}

return flowResponse;
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
    isMain,
    validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
    const options = {
        flowIdentifier: {
            type: "string",
            required: true,
        },
        flowAliasIdentifier: {
            type: "string",
            required: true,
        },
    },

```

```
    prompt: {
      type: "string",
    },
    region: {
      type: "string",
    },
  };
  const results = parseArgs({ options });
  const { errors } = validateArgs({ options }, results);
  return { errors, results };
};

if (isMain(import.meta.url)) {
  const { errors, results } = loadArgs();
  if (!errors) {
    invokeBedrockFlow(results.values);
  } else {
    console.error(errors.join("\n"));
  }
}
```

- Einzelheiten zur API finden Sie [InvokeFlow](#) in der AWS SDK für JavaScript API-Referenz.

CloudWatch Beispiele für die Verwendung von SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie AWS SDK für JavaScript (v3) mit verwenden CloudWatch.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kodex finden.

Themen

- [Aktionen](#)

Aktionen

DeleteAlarms

Das folgende Codebeispiel zeigt die Verwendung von `DeleteAlarms`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
import { DeleteAlarmsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new DeleteAlarmsCommand({
    AlarmNames: [process.env.CLOUDWATCH_ALARM_NAME], // Set the value of
    CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

Erstellen Sie den Client in einem separaten Modul und exportieren Sie ihn.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```


- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [DeleteAlarms](#) in der AWS SDK für JavaScript API-Referenz.

DescribeAlarmsForMetric

Das folgende Codebeispiel zeigt die Verwendung `DescribeAlarmsForMetric`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
import { DescribeAlarmsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new DescribeAlarmsCommand({
    AlarmNames: [process.env.CLOUDWATCH_ALARM_NAME], // Set the value of
    CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

Erstellen Sie den Client in einem separaten Modul und exportieren Sie ihn.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [DescribeAlarmsForMetric](#) in der AWS SDK für JavaScript API-Referenz.

DisableAlarmActions

Das folgende Codebeispiel zeigt die Verwendung `DisableAlarmActions`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
import { DisableAlarmActionsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new DisableAlarmActionsCommand({
    AlarmNames: process.env.CLOUDWATCH_ALARM_NAME, // Set the value of
    CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

Erstellen Sie den Client in einem separaten Modul und exportieren Sie ihn.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [DisableAlarmActions](#) in der AWS SDK für JavaScript API-Referenz.

EnableAlarmActions

Das folgende Codebeispiel zeigt die Verwendung `EnableAlarmActions`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
import { EnableAlarmActionsCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new EnableAlarmActionsCommand({
    AlarmNames: [process.env.CLOUDWATCH_ALARM_NAME], // Set the value of
    CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

Erstellen Sie den Client in einem separaten Modul und exportieren Sie ihn.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [EnableAlarmActions](#) in der AWS SDK für JavaScript API-Referenz.

ListMetrics

Das folgende Codebeispiel zeigt die Verwendung `ListMetrics`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
import {
  CloudWatchServiceException,
  ListMetricsCommand,
} from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

export const main = async () => {
  // Use the AWS console to see available namespaces and metric names. Custom
  // metrics can also be created.
  // https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/
  // viewing_metrics_with_cloudwatch.html
  const command = new ListMetricsCommand({
    Dimensions: [
      {
```

```
        Name: "LogGroupName",
      },
    ],
    MetricName: "IncomingLogEvents",
    Namespace: "AWS/Logs",
  });

  try {
    const response = await client.send(command);
    console.log(`Metrics count: ${response.Metrics?.length}`);
    return response;
  } catch (caught) {
    if (caught instanceof CloudWatchServiceException) {
      console.error(`Error from CloudWatch. ${caught.name}: ${caught.message}`);
    } else {
      throw caught;
    }
  }
};
```

Erstellen Sie den Client in einem separaten Modul und exportieren Sie ihn.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";

export const client = new CloudWatchClient({});
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [ListMetrics](#) in der AWS SDK für JavaScript API-Referenz.

PutMetricAlarm

Das folgende Codebeispiel zeigt die Verwendung `PutMetricAlarm`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
import { PutMetricAlarmCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  // This alarm triggers when CPUUtilization exceeds 70% for one minute.
  const command = new PutMetricAlarmCommand({
    AlarmName: process.env.CLOUDWATCH_ALARM_NAME, // Set the value of
    CLOUDWATCH_ALARM_NAME to the name of an existing alarm.
    ComparisonOperator: "GreaterThanThreshold",
    EvaluationPeriods: 1,
    MetricName: "CPUUtilization",
    Namespace: "AWS/EC2",
    Period: 60,
    Statistic: "Average",
    Threshold: 70.0,
    ActionsEnabled: false,
    AlarmDescription: "Alarm when server CPU exceeds 70%",
    Dimensions: [
      {
        Name: "InstanceId",
        Value: process.env.EC2_INSTANCE_ID, // Set the value of EC_INSTANCE_ID to
        the Id of an existing Amazon EC2 instance.
      },
    ],
    Unit: "Percent",
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

Erstellen Sie den Client in einem separaten Modul und exportieren Sie ihn.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";
```

```
export const client = new CloudWatchClient({});
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [PutMetricAlarm](#) in der AWS SDK für JavaScript API-Referenz.

PutMetricData

Das folgende Codebeispiel zeigt die Verwendung `PutMetricData`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
import { PutMetricDataCommand } from "@aws-sdk/client-cloudwatch";
import { client } from "../libs/client.js";

const run = async () => {
  // See https://docs.aws.amazon.com/AmazonCloudWatch/latest/APIReference/
  // API_PutMetricData.html#API_PutMetricData_RequestParameters
  // and https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/
  // publishingMetrics.html
  // for more information about the parameters in this command.
  const command = new PutMetricDataCommand({
    MetricData: [
      {
        MetricName: "PAGES_VISITED",
        Dimensions: [
          {
            Name: "UNIQUE_PAGES",
            Value: "URLS",
          },
        ],
        Unit: "None",
        Value: 1.0,
      },
    ],
  });
```

```
    },  
  ],  
  Namespace: "SITE/TRAFFIC",  
});  
  
try {  
  return await client.send(command);  
} catch (err) {  
  console.error(err);  
}  
};  
  
export default run();
```

Erstellen Sie den Client in einem separaten Modul und exportieren Sie ihn.

```
import { CloudWatchClient } from "@aws-sdk/client-cloudwatch";  
  
export const client = new CloudWatchClient({});
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [PutMetricData](#) in der AWS SDK für JavaScript API-Referenz.

CloudWatch Beispiele für Ereignisse mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie AWS SDK für JavaScript (v3) mit CloudWatch Ereignissen verwenden.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kodex finden.

Themen

- [Aktionen](#)

Aktionen

PutEvents

Das folgende Codebeispiel zeigt die Verwendung von `PutEvents`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
import { PutEventsCommand } from "@aws-sdk/client-cloudwatch-events";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new PutEventsCommand({
    // The list of events to send to Amazon CloudWatch Events.
    Entries: [
      {
        // The name of the application or service that is sending the event.
        Source: "my.app",

        // The name of the event that is being sent.
        DetailType: "My Custom Event",

        // The data that is sent with the event.
        Detail: JSON.stringify({ timeOfEvent: new Date().toISOString() }),
      },
    ],
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};
```

```
export default run();
```

Erstellen Sie den Client in einem separaten Modul und exportieren Sie ihn.

```
import { CloudWatchEventsClient } from "@aws-sdk/client-cloudwatch-events";  
  
export const client = new CloudWatchEventsClient({});
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [PutEvents](#) in der AWS SDK für JavaScript API-Referenz.

PutRule

Das folgende Codebeispiel zeigt die Verwendung PutRule.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel](#) einrichten und ausführen.

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
import { PutRuleCommand } from "@aws-sdk/client-cloudwatch-events";  
import { client } from "../libs/client.js";  
  
const run = async () => {  
  // Request parameters for PutRule.  
  // https://docs.aws.amazon.com/eventbridge/latest/APIReference/  
  API_PutRule.html#API_PutRule_RequestParameters  
  const command = new PutRuleCommand({  
    Name: process.env.CLOUDWATCH_EVENTS_RULE,  
  
    // The event pattern for the rule.  
    // Example: {"source": ["my.app"]}
```

```
    EventPattern: process.env.CLOUDWATCH_EVENTS_RULE_PATTERN,

    // The state of the rule. Valid values: ENABLED, DISABLED
    State: "ENABLED",
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

Erstellen Sie den Client in einem separaten Modul und exportieren Sie ihn.

```
import { CloudWatchEventsClient } from "@aws-sdk/client-cloudwatch-events";

export const client = new CloudWatchEventsClient({});
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [PutRule](#) in der AWS SDK für JavaScript API-Referenz.

PutTargets

Das folgende Codebeispiel zeigt die Verwendung `PutTargets`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel](#) einrichten und ausführen.

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
import { PutTargetsCommand } from "@aws-sdk/client-cloudwatch-events";
```

```
import { client } from "../libs/client.js";

const run = async () => {
  const command = new PutTargetsCommand({
    // The name of the Amazon CloudWatch Events rule.
    Rule: process.env.CLOUDWATCH_EVENTS_RULE,

    // The targets to add to the rule.
    Targets: [
      {
        Arn: process.env.CLOUDWATCH_EVENTS_TARGET_ARN,
        // The ID of the target. Choose a unique ID for each target.
        Id: process.env.CLOUDWATCH_EVENTS_TARGET_ID,
      },
    ],
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

Erstellen Sie den Client in einem separaten Modul und exportieren Sie ihn.

```
import { CloudWatchEventsClient } from "@aws-sdk/client-cloudwatch-events";

export const client = new CloudWatchEventsClient({});
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [PutTargets](#) in der AWS SDK für JavaScript API-Referenz.

CloudWatch Protokolliert Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie AWS SDK für JavaScript (v3) mit CloudWatch Logs verwenden.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien anzeigen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb eines Service aufrufen oder mit anderen AWS-Services kombinieren.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kodex finden.

Themen

- [Aktionen](#)
- [Szenarien](#)

Aktionen

CreateLogGroup

Das folgende Codebeispiel zeigt die Verwendung von `CreateLogGroup`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { CreateLogGroupCommand } from "@aws-sdk/client-cloudwatch-logs";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new CreateLogGroupCommand({
    // The name of the log group.
    logGroupName: process.env.CLOUDWATCH_LOGS_LOG_GROUP,
  });

  try {
    return await client.send(command);
  } catch (err) {
```

```
    console.error(err);
  }
};

export default run();
```

- Einzelheiten zur API finden Sie [CreateLogGroup](#) in der AWS SDK für JavaScript API-Referenz.

DeleteLogGroup

Das folgende Codebeispiel zeigt die Verwendung `DeleteLogGroup`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { DeleteLogGroupCommand } from "@aws-sdk/client-cloudwatch-logs";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new DeleteLogGroupCommand({
    // The name of the log group.
    logGroupName: process.env.CLOUDWATCH_LOGS_LOG_GROUP,
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

- Einzelheiten zur API finden Sie [DeleteLogGroup](#) in der AWS SDK für JavaScript API-Referenz.

DeleteSubscriptionFilter

Das folgende Codebeispiel zeigt die Verwendung `DeleteSubscriptionFilter`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { DeleteSubscriptionFilterCommand } from "@aws-sdk/client-cloudwatch-logs";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new DeleteSubscriptionFilterCommand({
    // The name of the filter.
    filterName: process.env.CLOUDWATCH_LOGS_FILTER_NAME,
    // The name of the log group.
    logGroupName: process.env.CLOUDWATCH_LOGS_LOG_GROUP,
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

- Einzelheiten zur API finden Sie [DeleteSubscriptionFilter](#) in der AWS SDK für JavaScript API-Referenz.

DescribeLogGroups

Das folgende Codebeispiel zeigt die Verwendung `DescribeLogGroups`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import {
  paginateDescribeLogGroups,
  CloudWatchLogsClient,
} from "@aws-sdk/client-cloudwatch-logs";

const client = new CloudWatchLogsClient({});

export const main = async () => {
  const paginatedLogGroups = paginateDescribeLogGroups({ client }, {});
  const logGroups = [];

  for await (const page of paginatedLogGroups) {
    if (page.logGroups?.every((lg) => !!lg)) {
      logGroups.push(...page.logGroups);
    }
  }

  console.log(logGroups);
  return logGroups;
};
```

- Einzelheiten zur API finden Sie [DescribeLogGroups](#) in der AWS SDK für JavaScript API-Referenz.

DescribeSubscriptionFilters

Das folgende Codebeispiel zeigt die Verwendung `DescribeSubscriptionFilters`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { DescribeSubscriptionFiltersCommand } from "@aws-sdk/client-cloudwatch-logs";
import { client } from "../libs/client.js";

const run = async () => {
  // This will return a list of all subscription filters in your account
  // matching the log group name.
  const command = new DescribeSubscriptionFiltersCommand({
    logGroupName: process.env.CLOUDWATCH_LOGS_LOG_GROUP,
    limit: 1,
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();
```

- Einzelheiten zur API finden Sie [DescribeSubscriptionFilters](#) in der AWS SDK für JavaScript API-Referenz.

GetQueryResults

Das folgende Codebeispiel zeigt die Verwendung `GetQueryResults`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
/**
 * Simple wrapper for the GetQueryResultsCommand.
 * @param {string} queryId
 */
_getQueryResults(queryId) {
  return this.client.send(new GetQueryResultsCommand({ queryId }));
}
```

- Einzelheiten zur API finden Sie [GetQueryResults](#) in der AWS SDK für JavaScript API-Referenz.

PutSubscriptionFilter

Das folgende Codebeispiel zeigt die Verwendung `PutSubscriptionFilter`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { PutSubscriptionFilterCommand } from "@aws-sdk/client-cloudwatch-logs";
import { client } from "../libs/client.js";

const run = async () => {
  const command = new PutSubscriptionFilterCommand({
    // An ARN of a same-account Kinesis stream, Kinesis Firehose
    // delivery stream, or Lambda function.
    // https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/
    SubscriptionFilters.html
  });
```

```

    destinationArn: process.env.CLOUDWATCH_LOGS_DESTINATION_ARN,

    // A name for the filter.
    filterName: process.env.CLOUDWATCH_LOGS_FILTER_NAME,

    // A filter pattern for subscribing to a filtered stream of log events.
    // https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/
    FilterAndPatternSyntax.html
    filterPattern: process.env.CLOUDWATCH_LOGS_FILTER_PATTERN,

    // The name of the log group. Messages in this group matching the filter pattern
    // will be sent to the destination ARN.
    logGroupName: process.env.CLOUDWATCH_LOGS_LOG_GROUP,
  });

  try {
    return await client.send(command);
  } catch (err) {
    console.error(err);
  }
};

export default run();

```

- Einzelheiten zur API finden Sie [PutSubscriptionFilter](#) in der AWS SDK für JavaScript API-Referenz.

StartLiveTail

Das folgende Codebeispiel zeigt die Verwendung `StartLiveTail`.

SDK für JavaScript (v3)

Binden Sie die erforderlichen Dateien ein.

```
import { CloudWatchLogsClient, StartLiveTailCommand } from "@aws-sdk/client-cloudwatch-logs";
```

Bearbeiten Sie die Ereignisse aus der Live-Tail-Sitzung.

```
async function handleResponseAsync(response) {
  try {
    for await (const event of response.responseStream) {
      if (event.sessionStart !== undefined) {
        console.log(event.sessionStart);
      } else if (event.sessionUpdate !== undefined) {
        for (const logEvent of event.sessionUpdate.sessionResults) {
          const timestamp = logEvent.timestamp;
          const date = new Date(timestamp);
          console.log "[" + date + "]" + logEvent.message);
        }
      } else {
        console.error("Unknown event type");
      }
    }
  } catch (err) {
    // On-stream exceptions are captured here
    console.error(err)
  }
}
```

Starten Sie die Live-Tail-Sitzung.

```
const client = new CloudWatchLogsClient();

const command = new StartLiveTailCommand({
  logGroupIdentifiers: logGroupIdentifiers,
  logStreamNames: logStreamNames,
  logEventFilterPattern: filterPattern
});

try{
  const response = await client.send(command);
  handleResponseAsync(response);
} catch (err){
  // Pre-stream exceptions are captured here
  console.log(err);
}
```

Beenden Sie die Live-Tail-Sitzung nach Ablauf einer gewissen Zeit.

```

/* Set a timeout to close the client. This will stop the Live Tail session. */
setTimeout(function() {
  console.log("Client timeout");
  client.destroy();
}, 10000);

```

- Einzelheiten zur API finden Sie [StartLiveTail](#) in der AWS SDK für JavaScript API-Referenz.

StartQuery

Das folgende Codebeispiel zeigt die Verwendung `StartQuery`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```

/**
 * Wrapper for the StartQueryCommand. Uses a static query string
 * for consistency.
 * @param {[Date, Date]} dateRange
 * @param {number} maxLogs
 * @returns {Promise<{ queryId: string }>}
 */
async _startQuery([startDate, endDate], maxLogs = 10000) {
  try {
    return await this.client.send(
      new StartQueryCommand({
        logGroupNames: this.logGroupNames,
        queryString: "fields @timestamp, @message | sort @timestamp asc",
        startTime: startDate.valueOf(),
        endTime: endDate.valueOf(),
        limit: maxLogs,
      }),
    );
  } catch (err) {
    /** @type {string} */

```

```
const message = err.message;
if (message.startsWith("Query's end date and time")) {
  // This error indicates that the query's start or end date occur
  // before the log group was created.
  throw new DateOutOfBoundsError(message);
}

throw err;
}
```

- Einzelheiten zur API finden Sie [StartQuery](#) in der AWS SDK für JavaScript API-Referenz.

Szenarien

Ausführen einer umfangreichen Abfrage

Das folgende Codebeispiel zeigt, wie CloudWatch Logs verwendet werden kann, um mehr als 10.000 Datensätze abzufragen.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Dies ist der Einstiegspunkt.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { CloudWatchLogsClient } from "@aws-sdk/client-cloudwatch-logs";
import { CloudWatchQuery } from "./cloud-watch-query.js";

console.log("Starting a recursive query...");

if (!process.env.QUERY_START_DATE || !process.env.QUERY_END_DATE) {
  throw new Error(
    "QUERY_START_DATE and QUERY_END_DATE environment variables are required.",
  );
}
```

```

    );
  }

  const cloudWatchQuery = new CloudWatchQuery(new CloudWatchLogsClient({}), {
    logGroupNames: ["/workflows/cloudwatch-logs/large-query"],
    dateRange: [
      new Date(Number.parseInt(process.env.QUERY_START_DATE)),
      new Date(Number.parseInt(process.env.QUERY_END_DATE)),
    ],
  });

  await cloudWatchQuery.run();

  console.log(
    `Queries finished in ${cloudWatchQuery.secondsElapsed} seconds.\nTotal logs found: ${cloudWatchQuery.results.length}`,
  );

```

Dies ist eine Klasse, die Abfragen bei Bedarf in mehrere Schritte aufteilt.

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  StartQueryCommand,
  GetQueryResultsCommand,
} from "@aws-sdk/client-cloudwatch-logs";
import { splitDateRange } from "@aws-doc-sdk-examples/lib/utils/util-date.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

class DateOutOfBoundsError extends Error {}

export class CloudWatchQuery {
  /**
   * Run a query for all CloudWatch Logs within a certain date range.
   * CloudWatch logs return a max of 10,000 results. This class
   * performs a binary search across all of the logs in the provided
   * date range if a query returns the maximum number of results.
   *
   * @param {import('@aws-sdk/client-cloudwatch-logs').CloudWatchLogsClient} client
   * @param {{ logGroupNames: string[], dateRange: [Date, Date], queryConfig:
   * { limit: number } }} config
   */

```

```

constructor(client, { logGroupNames, dateRange, queryConfig }) {
  this.client = client;
  /**
   * All log groups are queried.
   */
  this.logGroupNames = logGroupNames;

  /**
   * The inclusive date range that is queried.
   */
  this.dateRange = dateRange;

  /**
   * CloudWatch Logs never returns more than 10,000 logs.
   */
  this.limit = queryConfig?.limit ?? 10000;

  /**
   * @type {import("@aws-sdk/client-cloudwatch-logs").ResultField[][]}
   */
  this.results = [];
}

/**
 * Run the query.
 */
async run() {
  this.secondsElapsed = 0;
  const start = new Date();
  this.results = await this._largeQuery(this.dateRange);
  const end = new Date();
  this.secondsElapsed = (end - start) / 1000;
  return this.results;
}

/**
 * Recursively query for logs.
 * @param {[Date, Date]} dateRange
 * @returns {Promise<import("@aws-sdk/client-cloudwatch-logs").ResultField[][]>}
 */
async _largeQuery(dateRange) {
  const logs = await this._query(dateRange, this.limit);

  console.log(

```



```

    `Query date range: ${dateRange
      .map((d) => d.toISOString())
      .join(" to ")}. Found ${logs.length} logs.`
  );

  if (logs.length < this.limit) {
    return logs;
  }

  const lastLogDate = this._getLastLogDate(logs);
  const offsetLastLogDate = new Date(lastLogDate);
  offsetLastLogDate.setMilliseconds(lastLogDate.getMilliseconds() + 1);
  const subDateRange = [offsetLastLogDate, dateRange[1]];
  const [r1, r2] = splitDateRange(subDateRange);
  const results = await Promise.all([
    this._largeQuery(r1),
    this._largeQuery(r2),
  ]);
  return [logs, ...results].flat();
}

/**
 * Find the most recent log in a list of logs.
 * @param {import("@aws-sdk/client-cloudwatch-logs").ResultField[][]} logs
 */
_getLastLogDate(logs) {
  const timestamps = logs
    .map(
      (log) =>
        log.find((fieldMeta) => fieldMeta.field === "@timestamp")?.value,
    )
    .filter((t) => !!t)
    .map((t) => `${t}Z`)
    .sort();

  if (!timestamps.length) {
    throw new Error("No timestamp found in logs.");
  }

  return new Date(timestamps[timestamps.length - 1]);
}

/**
 * Simple wrapper for the GetQueryResultsCommand.

```

```

    * @param {string} queryId
    */
    _getQueryResults(queryId) {
        return this.client.send(new GetQueryResultsCommand({ queryId }));
    }

    /**
     * Starts a query and waits for it to complete.
     * @param {[Date, Date]} dateRange
     * @param {number} maxLogs
     */
    async _query(dateRange, maxLogs) {
        try {
            const { queryId } = await this._startQuery(dateRange, maxLogs);
            const { results } = await this._waitUntilQueryDone(queryId);
            return results ?? [];
        } catch (err) {
            /**
             * This error is thrown when StartQuery returns an error indicating
             * that the query's start or end date occur before the log group was
             * created.
             */
            if (err instanceof DateOutOfBoundsError) {
                return [];
            }
            throw err;
        }
    }

    /**
     * Wrapper for the StartQueryCommand. Uses a static query string
     * for consistency.
     * @param {[Date, Date]} dateRange
     * @param {number} maxLogs
     * @returns {Promise<{ queryId: string }>}
     */
    async _startQuery([startDate, endDate], maxLogs = 10000) {
        try {
            return await this.client.send(
                new StartQueryCommand({
                    logGroupNames: this.logGroupNames,
                    queryString: "fields @timestamp, @message | sort @timestamp asc",
                    startTime: startDate.valueOf(),
                    endTime: endDate.valueOf(),
                })
            );
        }
    }

```

```

        limit: maxLogs,
      )),
    );
  } catch (err) {
    /** @type {string} */
    const message = err.message;
    if (message.startsWith("Query's end date and time")) {
      // This error indicates that the query's start or end date occur
      // before the log group was created.
      throw new DateOutOfBoundsError(message);
    }

    throw err;
  }
}

/**
 * Call GetQueryResultsCommand until the query is done.
 * @param {string} queryId
 */
_waitUntilQueryDone(queryId) {
  const getResults = async () => {
    const results = await this._getQueryResults(queryId);
    const queryDone = [
      "Complete",
      "Failed",
      "Cancelled",
      "Timeout",
      "Unknown",
    ].includes(results.status);

    return { queryDone, results };
  };

  return retry(
    { intervalInMs: 1000, maxRetries: 60, quiet: true },
    async () => {
      const { queryDone, results } = await getResults();
      if (!queryDone) {
        throw new Error("Query not done.");
      }

      return results;
    },
  ),

```

```
    );  
  }  
}
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für JavaScript - API-Referenz.
 - [GetQueryResults](#)
 - [StartQuery](#)

Verwendung geplanter Ereignisse zum Aufrufen einer Lambda-Funktion

Das folgende Codebeispiel zeigt, wie eine AWS Lambda Funktion erstellt wird, die durch ein von Amazon EventBridge geplantes Ereignis aufgerufen wird.

SDK für JavaScript (v3)

Zeigt, wie ein von Amazon EventBridge geplantes Ereignis erstellt wird, das eine AWS Lambda Funktion aufruft. Konfigurieren Sie so EventBridge, dass ein Cron-Ausdruck verwendet wird, um zu planen, wann die Lambda-Funktion aufgerufen wird. In diesem Beispiel erstellen Sie eine Lambda-Funktion mithilfe der JavaScript Lambda-Laufzeit-API. In diesem Beispiel werden verschiedene AWS Dienste aufgerufen, um einen bestimmten Anwendungsfall auszuführen. Dieses Beispiel zeigt, wie man eine App erstellt, die eine mobile Textnachricht an Ihre Mitarbeiter sendet, um ihnen zum einjährigen Jubiläum zu gratulieren.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

Dieses Beispiel ist auch verfügbar im [AWS SDK für JavaScript Entwicklerhandbuch für v3](#).

In diesem Beispiel verwendete Dienste

- CloudWatch Logs
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

CodeBuild Beispiele für die Verwendung von SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie AWS SDK für JavaScript (v3) mit verwenden CodeBuild.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kodex finden.

Themen

- [Aktionen](#)

Aktionen

CreateProject

Das folgende Codebeispiel zeigt die Verwendung von `CreateProject`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Ein Projekt zu erstellen.

```
import {
  ArtifactsType,
  CodeBuildClient,
  ComputeType,
  CreateProjectCommand,
  EnvironmentType,
  SourceType,
} from "@aws-sdk/client-codebuild";
```

```

// Create the AWS CodeBuild project.
export const createProject = async (
  projectName = "MyCodeBuilder",
  roleArn = "arn:aws:iam::xxxxxxxxxxxx:role/CodeBuildAdmin",
  buildOutputBucket = "xxxx",
  githubUrl = "https://...",
) => {
  const codeBuildClient = new CodeBuildClient({});

  const response = await codeBuildClient.send(
    new CreateProjectCommand({
      artifacts: {
        // The destination of the build artifacts.
        type: ArtifactsType.S3,
        location: buildOutputBucket,
      },
      // Information about the build environment. The combination of "computeType"
and "type" determines the
      // requirements for the environment such as CPU, memory, and disk space.
      environment: {
        // Build environment compute types.
        // https://docs.aws.amazon.com/codebuild/latest/userguide/build-env-ref-
compute-types.html
        computeType: ComputeType.BUILD_GENERAL1_SMALL,
        // Docker image identifier.
        // See https://docs.aws.amazon.com/codebuild/latest/userguide/build-env-ref-
available.html
        image: "aws/codebuild/standard:7.0",
        // Build environment type.
        type: EnvironmentType.LINUX_CONTAINER,
      },
      name: projectName,
      // A role ARN with permission to create a CodeBuild project, write to the
artifact location, and write CloudWatch logs.
      serviceRole: roleArn,
      source: {
        // The type of repository that contains the source code to be built.
        type: SourceType.GITHUB,
        // The location of the repository that contains the source code to be built.
        location: githubUrl,
      },
    })),
  );
}

```

```
console.log(response);
// {
//   '$metadata': {
//     httpStatusCode: 200,
//     requestId: 'b428b244-777b-49a6-a48d-5dffedced8e7',
//     extendedRequestId: undefined,
//     cfId: undefined,
//     attempts: 1,
//     totalRetryDelay: 0
//   },
//   project: {
//     arn: 'arn:aws:codebuild:us-east-1:xxxxxxxxxxxx:project/MyCodeBuilder',
//     artifacts: {
//       encryptionDisabled: false,
//       location: 'xxxxxx-xxxxxx-xxxxxx',
//       name: 'MyCodeBuilder',
//       namespaceType: 'NONE',
//       packaging: 'NONE',
//       type: 'S3'
//     },
//     badge: { badgeEnabled: false },
//     cache: { type: 'NO_CACHE' },
//     created: 2023-08-18T14:46:48.979Z,
//     encryptionKey: 'arn:aws:kms:us-east-1:xxxxxxxxxxxx:alias/aws/s3',
//     environment: {
//       computeType: 'BUILD_GENERAL1_SMALL',
//       environmentVariables: [],
//       image: 'aws/codebuild/standard:7.0',
//       imagePullCredentialsType: 'CODEBUILD',
//       privilegedMode: false,
//       type: 'LINUX_CONTAINER'
//     },
//     lastModified: 2023-08-18T14:46:48.979Z,
//     name: 'MyCodeBuilder',
//     projectVisibility: 'PRIVATE',
//     queuedTimeoutInMinutes: 480,
//     serviceRole: 'arn:aws:iam:xxxxxxxxxxxx:role/CodeBuildAdmin',
//     source: {
//       insecureSsl: false,
//       location: 'https://...',
//       reportBuildStatus: false,
//       type: 'GITHUB'
//     },
//     timeoutInMinutes: 60
//   }
// }
```

```
//    }  
//  }  
  return response;  
};
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [CreateProject](#)in der AWS SDK für JavaScript API-Referenz.

Beispiele für Amazon Cognito Identity mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK für JavaScript (v3) mit Amazon Cognito Identity Aktionen ausführen und allgemeine Szenarien implementieren.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb eines Service aufrufen oder mit anderen AWS-Services kombinieren.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kodex finden.

Themen

- [Szenarien](#)

Szenarien

Erstellen Sie eine Amazon-Texttract-Explorer-Anwendung

Die folgenden Codebeispiele zeigen, wie die Amazon-Texttract-Ausgabe in einer interaktiven Anwendung untersucht werden kann.

SDK für JavaScript (v3)

Zeigt, wie Sie mit AWS SDK für JavaScript dem eine React-Anwendung erstellen, die Amazon Texttract verwendet, um Daten aus einem Dokumentbild zu extrahieren und auf einer interaktiven Webseite anzuzeigen. Dieses Beispiel wird in einem Webbrowser ausgeführt und erfordert eine authentifizierte Amazon-Cognito-Identität für Anmeldeinformationen. Es verwendet Amazon Simple Storage Service (Amazon S3) zur Speicherung und fragt für Benachrichtigungen eine Amazon Simple Queue Service (Amazon SQS)-Warteschlange ab, die ein Amazon Simple Notification Service (Amazon SNS)-Thema abonniert hat.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

In diesem Beispiel verwendete Dienste

- Amazon Cognito Identity
- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon Textract

Beispiele für Amazon Cognito Identity Provider mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie AWS SDK für JavaScript (v3) mit Amazon Cognito Identity Provider verwenden.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien anzeigen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb eines Service aufrufen oder mit anderen AWS-Services kombinieren.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kodex finden.

Themen

- [Erste Schritte](#)
- [Aktionen](#)
- [Szenarien](#)

Erste Schritte

Hello Amazon Cognito

Die folgenden Codebeispiele veranschaulichen die ersten Schritte mit Amazon Cognito.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import {
  paginateListUserPools,
  CognitoIdentityProviderClient,
} from "@aws-sdk/client-cognito-identity-provider";

const client = new CognitoIdentityProviderClient({});

export const helloCognito = async () => {
  const paginator = paginateListUserPools({ client }, {});

  const userPoolNames = [];

  for await (const page of paginator) {
    const names = page.UserPools.map((pool) => pool.Name);
    userPoolNames.push(...names);
  }

  console.log("User pool names: ");
  console.log(userPoolNames.join("\n"));
  return userPoolNames;
};
```

- Einzelheiten zur API finden Sie [ListUserPools](#) in der AWS SDK für JavaScript API-Referenz.

Aktionen

AdminGetUser

Das folgende Codebeispiel zeigt die Verwendung `AdminGetUser`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
const adminGetUser = ({ userPoolId, username }) => {  
  const client = new CognitoIdentityProviderClient({});  
  
  const command = new AdminGetUserCommand({  
    UserPoolId: userPoolId,  
    Username: username,  
  });  
  
  return client.send(command);  
};
```

- Einzelheiten zur API finden Sie [AdminGetUser](#) in der AWS SDK für JavaScript API-Referenz.

AdminInitiateAuth

Das folgende Codebeispiel zeigt die Verwendung `AdminInitiateAuth`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
const adminInitiateAuth = ({ clientId, userPoolId, username, password }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new AdminInitiateAuthCommand({
    ClientId: clientId,
    UserPoolId: userPoolId,
    AuthFlow: AuthFlowType.ADMIN_USER_PASSWORD_AUTH,
    AuthParameters: { USERNAME: username, PASSWORD: password },
  });

  return client.send(command);
};
```

- Einzelheiten zur API finden Sie [AdminInitiateAuth](#) in der AWS SDK für JavaScript API-Referenz.

AdminRespondToAuthChallenge

Das folgende Codebeispiel zeigt die Verwendung `AdminRespondToAuthChallenge`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
const adminRespondToAuthChallenge = ({
  userPoolId,
  clientId,
  username,
  totp,
  session,
}) => {
  const client = new CognitoIdentityProviderClient({});
  const command = new AdminRespondToAuthChallengeCommand({
    ChallengeName: ChallengeNameType.SOFTWARE_TOKEN_MFA,
    ChallengeResponses: {
      SOFTWARE_TOKEN_MFA_CODE: totp,
      USERNAME: username,
    },
  });
```

```
    },
    ClientId: clientId,
    UserPoolId: userPoolId,
    Session: session,
  });

  return client.send(command);
};
```

- Einzelheiten zur API finden Sie [AdminRespondToAuthChallenge](#) in der AWS SDK für JavaScript API-Referenz.

AssociateSoftwareToken

Das folgende Codebeispiel zeigt die Verwendung `AssociateSoftwareToken`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
const associateSoftwareToken = (session) => {
  const client = new CognitoIdentityProviderClient({});
  const command = new AssociateSoftwareTokenCommand({
    Session: session,
  });

  return client.send(command);
};
```

- Einzelheiten zur API finden Sie [AssociateSoftwareToken](#) in der AWS SDK für JavaScript API-Referenz.

ConfirmDevice

Das folgende Codebeispiel zeigt die Verwendung `ConfirmDevice`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
const confirmDevice = ({ deviceKey, accessToken, passwordVerifier, salt }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new ConfirmDeviceCommand({
    DeviceKey: deviceKey,
    AccessToken: accessToken,
    DeviceSecretVerifierConfig: {
      PasswordVerifier: passwordVerifier,
      Salt: salt,
    },
  });

  return client.send(command);
};
```

- Einzelheiten zur API finden Sie [ConfirmDevice](#) in der AWS SDK für JavaScript API-Referenz.

ConfirmSignUp

Das folgende Codebeispiel zeigt die Verwendung `ConfirmSignUp`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
const confirmSignUp = ({ clientId, username, code }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new ConfirmSignUpCommand({
    ClientId: clientId,
    Username: username,
    ConfirmationCode: code,
  });

  return client.send(command);
};
```

- Einzelheiten zur API finden Sie [ConfirmSignUp](#) in der AWS SDK für JavaScript API-Referenz.

DeleteUser

Das folgende Codebeispiel zeigt die Verwendung `DeleteUser`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel](#) einrichten und ausführen.

```
/**
 * Delete the signed-in user. Useful for allowing a user to delete their
 * own profile.
 * @param {{ region: string, accessToken: string }} config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-provider").DeleteUserCommandOutput | null, unknown]>}
 */
export const deleteUser = async ({ region, accessToken }) => {
  try {
    const client = new CognitoIdentityProviderClient({ region });
    const response = await client.send(
      new DeleteUserCommand({ AccessToken: accessToken }),
    );
    return [response, null];
  }
};
```

```
    } catch (err) {  
      return [null, err];  
    }  
  };  
};
```

- Einzelheiten zur API finden Sie [DeleteUser](#) in der AWS SDK für JavaScript API-Referenz.

InitiateAuth

Das folgende Codebeispiel zeigt die Verwendung `InitiateAuth`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
const initiateAuth = ({ username, password, clientId }) => {  
  const client = new CognitoIdentityProviderClient({});  
  
  const command = new InitiateAuthCommand({  
    AuthFlow: AuthFlowType.USER_PASSWORD_AUTH,  
    AuthParameters: {  
      USERNAME: username,  
      PASSWORD: password,  
    },  
    ClientId: clientId,  
  });  
  
  return client.send(command);  
};
```

- Einzelheiten zur API finden Sie [InitiateAuth](#) in der AWS SDK für JavaScript API-Referenz.

ListUsers

Das folgende Codebeispiel zeigt die Verwendung `ListUsers`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
const listUsers = ({ userPoolId }) => {  
  const client = new CognitoIdentityProviderClient({});  
  
  const command = new ListUsersCommand({  
    UserPoolId: userPoolId,  
  });  
  
  return client.send(command);  
};
```

- Einzelheiten zur API finden Sie [ListUsers](#) in der AWS SDK für JavaScript API-Referenz.

ResendConfirmationCode

Das folgende Codebeispiel zeigt die Verwendung `ResendConfirmationCode`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
const resendConfirmationCode = ({ clientId, username }) => {  
  const client = new CognitoIdentityProviderClient({});  
  
  const command = new ResendConfirmationCodeCommand({  
    ClientId: clientId,  
    Username: username,  
  });  
};
```

```
    return client.send(command);  
};
```

- Einzelheiten zur API finden Sie [ResendConfirmationCode](#) in der AWS SDK für JavaScript API-Referenz.

RespondToAuthChallenge

Das folgende Codebeispiel zeigt die Verwendung `RespondToAuthChallenge`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
const respondToAuthChallenge = ({  
  clientId,  
  username,  
  session,  
  userPoolId,  
  code,  
}) => {  
  const client = new CognitoIdentityProviderClient({});  
  
  const command = new RespondToAuthChallengeCommand({  
    ChallengeName: ChallengeNameType.SOFTWARE_TOKEN_MFA,  
    ChallengeResponses: {  
      SOFTWARE_TOKEN_MFA_CODE: code,  
      USERNAME: username,  
    },  
    ClientId: clientId,  
    UserPoolId: userPoolId,  
    Session: session,  
  });  
  
  return client.send(command);  
};
```

```
};
```

- Einzelheiten zur API finden Sie [RespondToAuthChallenge](#) in der AWS SDK für JavaScript API-Referenz.

SignUp

Das folgende Codebeispiel zeigt die Verwendung `SignUp`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
const signUp = ({ clientId, username, password, email }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new SignUpCommand({
    ClientId: clientId,
    Username: username,
    Password: password,
    UserAttributes: [{ Name: "email", Value: email }],
  });

  return client.send(command);
};
```

- Einzelheiten zur API finden Sie [SignUp](#) in der AWS SDK für JavaScript API-Referenz.

UpdateUserPool

Das folgende Codebeispiel zeigt die Verwendung `UpdateUserPool`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
/**
 * Connect a Lambda function to the PreSignUp trigger for a Cognito user pool
 * @param {{ region: string, userPoolId: string, handlerArn: string }} config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-
provider").UpdateUserPoolCommandOutput | null, unknown]>}
 */
export const addPreSignUpHandler = async ({
  region,
  userPoolId,
  handlerArn,
}) => {
  try {
    const cognitoClient = new CognitoIdentityProviderClient({
      region,
    });

    const command = new UpdateUserPoolCommand({
      UserPoolId: userPoolId,
      LambdaConfig: {
        PreSignUp: handlerArn,
      },
    });

    const response = await cognitoClient.send(command);
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};
```

- Einzelheiten zur API finden Sie [UpdateUserPool](#) in der AWS SDK für JavaScript API-Referenz.

VerifySoftwareToken

Das folgende Codebeispiel zeigt die Verwendung `VerifySoftwareToken`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
const verifySoftwareToken = (totp) => {
  const client = new CognitoIdentityProviderClient({});

  // The 'Session' is provided in the response to 'AssociateSoftwareToken'.
  const session = process.env.SESSION;

  if (!session) {
    throw new Error(
      "Missing a valid Session. Did you run 'admin-initiate-auth'?",
    );
  }

  const command = new VerifySoftwareTokenCommand({
    Session: session,
    UserCode: totp,
  });

  return client.send(command);
};
```

- Einzelheiten zur API finden Sie [VerifySoftwareToken](#) in der AWS SDK für JavaScript API-Referenz.

Szenarien

Bestätigen Sie automatisch bekannte Benutzer mit einer Lambda-Funktion

Die folgenden Codebeispiele zeigen, wie bekannte Amazon-Cognito-Benutzer mit einer Lambda-Funktion automatisch bestätigt werden können.

- Konfigurieren Sie einen Benutzerpool, um eine Lambda-Funktion für den PreSignUp-Trigger aufzurufen.
- Registrieren eines Benutzers bei Amazon Cognito.
- Die Lambda-Funktion scannt eine DynamoDB-Tabelle und bestätigt automatisch bekannte Benutzer.
- Melden Sie sich als neuer Benutzer an und bereinigen Sie dann die Ressourcen.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Konfigurieren Sie einen interaktiven „Szenario“-Lauf. Die JavaScript (v3) -Beispiele verwenden gemeinsam einen Szenario-Runner, um komplexe Beispiele zu vereinfachen. Der komplette Quellcode ist aktiviert GitHub.

```
import { AutoConfirm } from "../scenario-auto-confirm.js";

/**
 * The context is passed to every scenario. Scenario steps
 * will modify the context.
 */
const context = {
  errors: [],
  users: [
    {
      UserName: "test_user_1",
      UserEmail: "test_email_1@example.com",
    },
  ],
}
```

```

    {
      UserName: "test_user_2",
      userEmail: "test_email_2@example.com",
    },
    {
      UserName: "test_user_3",
      userEmail: "test_email_3@example.com",
    },
  ],
};

/**
 * Three Scenarios are created for the workflow. A Scenario is an orchestration
 * class
 * that simplifies running a series of steps.
 */
export const scenarios = {
  // Demonstrate automatically confirming known users in a database.
  "auto-confirm": AutoConfirm(context),
};

// Call function if run directly
import { fileURLToPath } from "node:url";
import { parseScenarioArgs } from "@aws-doc-sdk-examples/lib/scenario/index.js";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios, {
    name: "Cognito user pools and triggers",
    description:
      "Demonstrate how to use the AWS SDKs to customize Amazon Cognito
      authentication behavior.",
  });
}

```

Dieses Szenario demonstriert die automatische Bestätigung eines bekannten Benutzers. Es orchestriert die Beispielschritte.

```

import { wait } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";
import {
  Scenario,
  ScenarioAction,
  ScenarioInput,

```

```

    ScenarioOutput,
  } from "@aws-doc-sdk-examples/lib/scenario/scenario.js";

import {
  getStackOutputs,
  logCleanUpReminder,
  promptForStackName,
  promptForStackRegion,
  skipWhenErrors,
} from "./steps-common.js";
import { populateTable } from "./actions/dynamodb-actions.js";
import {
  addPreSignUpHandler,
  deleteUser,
  getUser,
  signIn,
  signUpUser,
} from "./actions/cognito-actions.js";
import {
  getLatestLogStreamForLambda,
  getLogEvents,
} from "./actions/cloudwatch-logs-actions.js";

/**
 * @typedef {{
 *   errors: Error[],
 *   password: string,
 *   users: { UserName: string, UserEmail: string }[],
 *   selectedUser?: string,
 *   stackName?: string,
 *   stackRegion?: string,
 *   token?: string,
 *   confirmDeleteSignedInUser?: boolean,
 *   TableName?: string,
 *   UserPoolClientId?: string,
 *   UserPoolId?: string,
 *   UserPoolArn?: string,
 *   AutoConfirmHandlerArn?: string,
 *   AutoConfirmHandlerName?: string
 * }} State
 */

const greeting = new ScenarioOutput(
  "greeting",

```



```
(/** @type {State} */ state) => `This demo will populate some users into the \
database created as part of the "${state.stackName}" stack. \
Then the AutoConfirmHandler will be linked to the PreSignUp \
trigger from Cognito. Finally, you will choose a user to sign up.`,\
  { skipWhen: skipWhenErrors },\
);\

const logPopulatingUsers = new ScenarioOutput(\
  "logPopulatingUsers",\
  "Populating the DynamoDB table with some users.",\
  { skipWhenErrors: skipWhenErrors },\
);\

const logPopulatingUsersComplete = new ScenarioOutput(\
  "logPopulatingUsersComplete",\
  "Done populating users.",\
  { skipWhen: skipWhenErrors },\
);\

const populateUsers = new ScenarioAction(\
  "populateUsers",\
  async (/** @type {State} */ state) => {\
    const [_, err] = await populateTable({\
      region: state.stackRegion,\
      tableName: state.TableName,\
      items: state.users,\
    });\
    if (err) {\
      state.errors.push(err);\
    }\
  },\
  {\
    skipWhen: skipWhenErrors,\
  },\
);\

const logSetupSignUpTrigger = new ScenarioOutput(\
  "logSetupSignUpTrigger",\
  "Setting up the PreSignUp trigger for the Cognito User Pool.",\
  { skipWhen: skipWhenErrors },\
);\

const setupSignUpTrigger = new ScenarioAction(\
  "setupSignUpTrigger",
```

```
async (** @type {State} */ state) => {
  const [, err] = await addPreSignUpHandler({
    region: state.stackRegion,
    userPoolId: state.UserPoolId,
    handlerArn: state.AutoConfirmHandlerArn,
  });
  if (err) {
    state.errors.push(err);
  }
},
{
  skipWhen: skipWhenErrors,
},
);

const logSetupSignUpTriggerComplete = new ScenarioOutput(
  "logSetupSignUpTriggerComplete",
  (
    /** @type {State} */ state,
  ) => `The lambda function "${state.AutoConfirmHandlerName}" \
has been configured as the PreSignUp trigger handler for the user pool
"${state.UserPoolId}".`,
  { skipWhen: skipWhenErrors },
);

const selectUser = new ScenarioInput(
  "selectedUser",
  "Select a user to sign up.",
  {
    type: "select",
    choices: (** @type {State} */ state) => state.users.map((u) => u.UserName),
    skipWhen: skipWhenErrors,
    default: (** @type {State} */ state) => state.users[0].UserName,
  },
);

const checkIfUserAlreadyExists = new ScenarioAction(
  "checkIfUserAlreadyExists",
  async (** @type {State} */ state) => {
    const [user, err] = await getUser({
      region: state.stackRegion,
      userPoolId: state.UserPoolId,
      username: state.selectedUser,
    });
  });
```

```

    if (err?.name === "UserNotFoundException") {
      // Do nothing. We're not expecting the user to exist before
      // sign up is complete.
      return;
    }

    if (err) {
      state.errors.push(err);
      return;
    }

    if (user) {
      state.errors.push(
        new Error(
          `The user "${state.selectedUser}" already exists in the user pool
"${state.UserPoolId}".`,
        ),
      );
    }
  },
  {
    skipWhen: skipWhenErrors,
  },
);

const createPassword = new ScenarioInput(
  "password",
  "Enter a password that has at least eight characters, uppercase, lowercase,
  numbers and symbols.",
  { type: "password", skipWhen: skipWhenErrors, default: "Abcd1234!" },
);

const logSignUpExistingUser = new ScenarioOutput(
  "logSignUpExistingUser",
  (/** @type {State} */ state) => `Signing up user "${state.selectedUser}".`,
  { skipWhen: skipWhenErrors },
);

const signUpExistingUser = new ScenarioAction(
  "signUpExistingUser",
  async (/** @type {State} */ state) => {
    const signUp = (password) =>
      signUpUser({

```

```
    region: state.stackRegion,
    userPoolClientId: state.UserPoolClientId,
    username: state.selectedUser,
    email: state.users.find((u) => u.UserName === state.selectedUser)
      .UserEmail,
    password,
  });

let [_, err] = await signUp(state.password);

while (err?.name === "InvalidPasswordException") {
  console.warn("The password you entered was invalid.");
  await createPassword.handle(state);
  [_, err] = await signUp(state.password);
}

if (err) {
  state.errors.push(err);
},
{ skipWhen: skipWhenErrors },
);

const logSignUpExistingUserComplete = new ScenarioOutput(
  "logSignUpExistingUserComplete",
  (/** @type {State} */ state) =>
    `"${state.selectedUser}" was signed up successfully.`,
  { skipWhen: skipWhenErrors },
);

const logLambdaLogs = new ScenarioAction(
  "logLambdaLogs",
  async (/** @type {State} */ state) => {
    console.log(
      "Waiting a few seconds to let Lambda write to CloudWatch Logs...\n",
    );
    await wait(10);

    const [logStream, logStreamErr] = await getLatestLogStreamForLambda({
      functionName: state.AutoConfirmHandlerName,
      region: state.stackRegion,
    });
    if (logStreamErr) {
      state.errors.push(logStreamErr);
    }
  }
);
```

```
    return;
  }

  console.log(
    `Getting some recent events from log stream "${logStream.logStreamName}"`,
  );
  const [logEvents, logEventsErr] = await getLogEvents({
    functionName: state.AutoConfirmHandlerName,
    region: state.stackRegion,
    eventCount: 10,
    logStreamName: logStream.logStreamName,
  });
  if (logEventsErr) {
    state.errors.push(logEventsErr);
    return;
  }

  console.log(logEvents.map((ev) => `t${ev.message}`).join(""));
},
{ skipWhen: skipWhenErrors },
);

const logSignInUser = new ScenarioOutput(
  "logSignInUser",
  (/** @type {State} */ state) => `Let's sign in as ${state.selectedUser}`,
  { skipWhen: skipWhenErrors },
);

const signInUser = new ScenarioAction(
  "signInUser",
  async (/** @type {State} */ state) => {
    const [response, err] = await signIn({
      region: state.stackRegion,
      clientId: state.UserPoolClientId,
      username: state.selectedUser,
      password: state.password,
    });
  });

  if (err?.name === "PasswordResetRequiredException") {
    state.errors.push(new Error("Please reset your password."));
    return;
  }

  if (err) {
```

```
        state.errors.push(err);
        return;
    }

    state.token = response?.AuthenticationResult?.AccessToken;
},
{ skipWhen: skipWhenErrors },
);

const logSignInUserComplete = new ScenarioOutput(
    "logSignInUserComplete",
    (/** @type {State} */ state) =>
        `Successfully signed in. Your access token starts with: ${state.token.slice(0,
11)}`,
    { skipWhen: skipWhenErrors },
);

const confirmDeleteSignedInUser = new ScenarioInput(
    "confirmDeleteSignedInUser",
    "Do you want to delete the currently signed in user?",
    { type: "confirm", skipWhen: skipWhenErrors },
);

const deleteSignedInUser = new ScenarioAction(
    "deleteSignedInUser",
    async (/** @type {State} */ state) => {
        const [_, err] = await deleteUser({
            region: state.stackRegion,
            accessToken: state.token,
        });

        if (err) {
            state.errors.push(err);
        }
    },
    {
        skipWhen: (/** @type {State} */ state) =>
            skipWhenErrors(state) || !state.confirmDeleteSignedInUser,
    },
);

const logErrors = new ScenarioOutput(
    "logErrors",
    (/** @type {State} */ state) => {
```

```
const errorList = state.errors
  .map((err) => ` - ${err.name}: ${err.message}`)
  .join("\n");
return `Scenario errors found:\n${errorList}`;
},
{
  // Don't log errors when there aren't any!
  skipWhen: (** @type {State} */ state) => state.errors.length === 0,
},
);

export const AutoConfirm = (context) =>
  new Scenario(
    "AutoConfirm",
    [
      promptForStackName,
      promptForStackRegion,
      getStackOutputs,
      greeting,
      logPopulatingUsers,
      populateUsers,
      logPopulatingUsersComplete,
      logSetupSignUpTrigger,
      setupSignUpTrigger,
      logSetupSignUpTriggerComplete,
      selectUser,
      checkIfUserAlreadyExists,
      createPassword,
      logSignUpExistingUser,
      signUpExistingUser,
      logSignUpExistingUserComplete,
      logLambdaLogs,
      logSignInUser,
      signInUser,
      logSignInUserComplete,
      confirmDeleteSignedInUser,
      deleteSignedInUser,
      logCleanUpReminder,
      logErrors,
    ],
    context,
  );
```

Dies sind Schritte, die mit anderen Szenarien geteilt werden.

```
import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/scenario.js";
import { getCfnOutputs } from "@aws-doc-sdk-examples/lib/sdk/cfn-outputs.js";

export const skipWhenErrors = (state) => state.errors.length > 0;

export const getStackOutputs = new ScenarioAction(
  "getStackOutputs",
  async (state) => {
    if (!state.stackName || !state.stackRegion) {
      state.errors.push(
        new Error(
          "No stack name or region provided. The stack name and \
region are required to fetch CFN outputs relevant to this example.",
        ),
      );
      return;
    }

    const outputs = await getCfnOutputs(state.stackName, state.stackRegion);
    Object.assign(state, outputs);
  },
);

export const promptForStackName = new ScenarioInput(
  "stackName",
  "Enter the name of the stack you deployed earlier.",
  { type: "input", default: "PoolsAndTriggersStack" },
);

export const promptForStackRegion = new ScenarioInput(
  "stackRegion",
  "Enter the region of the stack you deployed earlier.",
  { type: "input", default: "us-east-1" },
);

export const logCleanUpReminder = new ScenarioOutput(
  "logCleanUpReminder",
  "All done. Remember to run 'cdk destroy' to teardown the stack.",
);
```



```
{ skipWhen: skipWhenErrors },  
);
```

Ein Handler für den PreSignUp-Trigger mit einer Lambda-Funktion.

```
import type { PreSignUpTriggerEvent, Handler } from "aws-lambda";  
import type { UserRepository } from "../user-repository";  
import { DynamoDBUserRepository } from "../user-repository";  
  
export class PreSignUpHandler {  
  private userRepository: UserRepository;  
  
  constructor(userRepository: UserRepository) {  
    this.userRepository = userRepository;  
  }  
  
  private isPreSignUpTriggerSource(event: PreSignUpTriggerEvent): boolean {  
    return event.triggerSource === "PreSignUp_SignUp";  
  }  
  
  private getEventUserEmail(event: PreSignUpTriggerEvent): string {  
    return event.request.userAttributes.email;  
  }  
  
  async handlePreSignUpTriggerEvent(  
    event: PreSignUpTriggerEvent,  
  ): Promise<PreSignUpTriggerEvent> {  
    console.log(  
      `Received presignup from ${event.triggerSource} for user '${event.userName}'`,  
    );  
  
    if (!this.isPreSignUpTriggerSource(event)) {  
      return event;  
    }  
  
    const eventEmail = this.getEventUserEmail(event);  
    console.log(`Looking up email ${eventEmail}.`);  
    const storedUserInfo =  
      await this.userRepository.getUserInfoByEmail(eventEmail);  
  
    if (!storedUserInfo) {  
      console.log(  
        `User with email ${eventEmail} does not exist. Creating new user.  
        `);  
      const newUser = {  
        email: eventEmail,  
        password: event.password,  
        username: event.userName,  
      };  
      await this.userRepository.createUser(newUser);  
      console.log(`New user created with email ${eventEmail}.`);  
    }  
  }  
}
```

```

        `Email ${eventEmail} not found. Email verification is required.` ,
    );
    return event;
}

if (storedUserInfo.UserName !== event.userName) {
    console.log(
        `UserEmail ${eventEmail} found, but stored UserName
'${storedUserInfo.UserName}' does not match supplied UserName '${event.userName}'.
Verification is required.` ,
    );
} else {
    console.log(
        `UserEmail ${eventEmail} found with matching UserName
${storedUserInfo.UserName}. User is confirmed.` ,
    );
    event.response.autoConfirmUser = true;
    event.response.autoVerifyEmail = true;
}
return event;
}
}

const createPreSignUpHandler = (): PreSignUpHandler => {
    const tableName = process.env.TABLE_NAME;
    if (!tableName) {
        throw new Error("TABLE_NAME environment variable is not set");
    }

    const userRepository = new DynamoDBUserRepository(tableName);
    return new PreSignUpHandler(userRepository);
};

export const handler: Handler = async (event: PreSignUpTriggerEvent) => {
    const preSignUpHandler = createPreSignUpHandler();
    return preSignUpHandler.handlePreSignUpTriggerEvent(event);
};

```

Modul für CloudWatch Logs-Aktionen.

```
import {
```

```

    CloudWatchLogsClient,
    GetLogEventsCommand,
    OrderBy,
    paginateDescribeLogStreams,
  } from "@aws-sdk/client-cloudwatch-logs";

/**
 * Get the latest log stream for a Lambda function.
 * @param {{ functionName: string, region: string }} config
 * @returns {Promise<[import("@aws-sdk/client-cloudwatch-logs").LogStream | null,
  unknown]>}
 */
export const getLatestLogStreamForLambda = async ({ functionName, region }) => {
  try {
    const logGroupName = `/aws/lambda/${functionName}`;
    const cwlClient = new CloudWatchLogsClient({ region });
    const paginator = paginateDescribeLogStreams(
      { client: cwlClient },
      {
        descending: true,
        limit: 1,
        orderBy: OrderBy.LastEventTime,
        logGroupName,
      },
    );

    for await (const page of paginator) {
      return [page.logStreams[0], null];
    }
  } catch (err) {
    return [null, err];
  }
};

/**
 * Get the log events for a Lambda function's log stream.
 * @param {{
 *   functionName: string,
 *   logStreamName: string,
 *   eventCount: number,
 *   region: string
 * }} config
 * @returns {Promise<[import("@aws-sdk/client-cloudwatch-logs").OutputLogEvent[] |
  null, unknown]>}

```

```

*/
export const getLogEvents = async ({
  functionName,
  logStreamName,
  eventCount,
  region,
}) => {
  try {
    const cwlClient = new CloudWatchLogsClient({ region });
    const logGroupName = `/aws/lambda/${functionName}`;
    const response = await cwlClient.send(
      new GetLogEventsCommand({
        logStreamName: logStreamName,
        limit: eventCount,
        logGroupName: logGroupName,
      }),
    );

    return [response.events, null];
  } catch (err) {
    return [null, err];
  }
};

```

Modul für Amazon-Cognito-Aktionen.

```

import {
  AdminGetUserCommand,
  CognitoIdentityProviderClient,
  DeleteUserCommand,
  InitiateAuthCommand,
  SignUpCommand,
  UpdateUserPoolCommand,
} from "@aws-sdk/client-cognito-identity-provider";

/**
 * Connect a Lambda function to the PreSignUp trigger for a Cognito user pool
 * @param {{ region: string, userPoolId: string, handlerArn: string }} config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-provider").UpdateUserPoolCommandOutput | null, unknown]>}
 */

```

```
export const addPreSignUpHandler = async ({
  region,
  userPoolId,
  handlerArn,
}) => {
  try {
    const cognitoClient = new CognitoIdentityProviderClient({
      region,
    });

    const command = new UpdateUserPoolCommand({
      UserPoolId: userPoolId,
      LambdaConfig: {
        PreSignUp: handlerArn,
      },
    });

    const response = await cognitoClient.send(command);
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};

/**
 * Attempt to register a user to a user pool with a given username and password.
 * @param {{
 *   region: string,
 *   userPoolClientId: string,
 *   username: string,
 *   email: string,
 *   password: string
 * }} config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-provider").SignUpCommandOutput | null, unknown]>}
 */
export const signUpUser = async ({
  region,
  userPoolClientId,
  username,
  email,
  password,
}) => {
  try {
```

```
const cognitoClient = new CognitoIdentityProviderClient({
  region,
});

const response = await cognitoClient.send(
  new SignUpCommand({
    ClientId: userPoolClientId,
    Username: username,
    Password: password,
    UserAttributes: [{ Name: "email", Value: email }],
  }),
);
return [response, null];
} catch (err) {
  return [null, err];
}
};

/**
 * Sign in a user to Amazon Cognito using a username and password authentication
 * flow.
 * @param {{ region: string, clientId: string, username: string, password: string }}
 * config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-
 * provider").InitiateAuthCommandOutput | null, unknown]>}
 */
export const signIn = async ({ region, clientId, username, password }) => {
  try {
    const cognitoClient = new CognitoIdentityProviderClient({ region });
    const response = await cognitoClient.send(
      new InitiateAuthCommand({
        AuthFlow: "USER_PASSWORD_AUTH",
        ClientId: clientId,
        AuthParameters: { USERNAME: username, PASSWORD: password },
      }),
    );
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};

/**
 * Retrieve an existing user from a user pool.
```

```

* @param {{ region: string, userPoolId: string, username: string }} config
* @returns {Promise<[import("@aws-sdk/client-cognito-identity-provider").AdminGetUserCommandOutput | null, unknown]>}}
*/
export const getUser = async ({ region, userPoolId, username }) => {
  try {
    const cognitoClient = new CognitoIdentityProviderClient({ region });
    const response = await cognitoClient.send(
      new AdminGetUserCommand({
        UserPoolId: userPoolId,
        Username: username,
      }),
    );
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};

/**
* Delete the signed-in user. Useful for allowing a user to delete their
* own profile.
* @param {{ region: string, accessToken: string }} config
* @returns {Promise<[import("@aws-sdk/client-cognito-identity-provider").DeleteUserCommandOutput | null, unknown]>}}
*/
export const deleteUser = async ({ region, accessToken }) => {
  try {
    const client = new CognitoIdentityProviderClient({ region });
    const response = await client.send(
      new DeleteUserCommand({ AccessToken: accessToken }),
    );
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};

```

Modul für DynamoDB-Aktionen.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
```

```
import {
  BatchWriteCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

/**
 * Populate a DynamoDB table with provide items.
 * @param {{ region: string, tableName: string, items: Record<string, unknown>[] }}
  config
 * @returns {Promise<[import("@aws-sdk/lib-dynamodb").BatchWriteCommandOutput |
  null, unknown]>}
 */
export const populateTable = async ({ region, tableName, items }) => {
  try {
    const ddbClient = new DynamoDBClient({ region });
    const docClient = DynamoDBDocumentClient.from(ddbClient);
    const response = await docClient.send(
      new BatchWriteCommand({
        RequestItems: {
          [tableName]: items.map((item) => ({
            PutRequest: {
              Item: item,
            },
          })),
        },
      }),
    );
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für JavaScript - API-Referenz.
 - [DeleteUser](#)
 - [InitiateAuth](#)
 - [SignUp](#)
 - [UpdateUserPool](#)

Registrieren eines Benutzers bei einem Benutzerpool, der MFA erfordert

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Registrieren Sie einen Benutzer mit einem Benutzernamen, einem Passwort und einer E-Mail-Adresse und bestätigen Sie ihn.
- Richten Sie die Multi-Faktor-Authentifizierung durch Zuordnung einer MFA-Anwendung zu dem Benutzer ein.
- Melden Sie sich unter Verwendung eines Passworts und eines MFA-Codes an.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Um die beste Erfahrung zu erzielen, klonen Sie das GitHub Repository und führen Sie dieses Beispiel aus. Der folgende Code ist ein Teil der vollständigen Beispielanwendung.

```
import { logger } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { signUp } from "../../actions/sign-up.js";
import { FILE_USER_POOLS } from "../constants.js";
import { getSecondValuesFromEntries } from "@aws-doc-sdk-examples/lib/utils/util-csv.js";

const validateClient = (clientId) => {
  if (!clientId) {
    throw new Error(
      `App client id is missing. Did you run 'create-user-pool'?`,
    );
  }
};

const validateUser = (username, password, email) => {
  if (!(username && password && email)) {
    throw new Error(
      `Username, password, and email must be provided as arguments to the 'sign-up' command.`,
    );
  }
};
```

```

    }
  };

const signUpHandler = async (commands) => {
  const [_ , username, password, email] = commands;

  try {
    validateUser(username, password, email);
    /**
     * @type {string[]}
     */
    const values = getSecondValuesFromEntries(FILE_USER_POOLS);
    const clientId = values[0];
    validateClient(clientId);
    logger.log("Signing up.");
    await signUp({ clientId, username, password, email });
    logger.log(`Signed up. A confirmation email has been sent to: ${email}.`);
    logger.log(
      `Run 'confirm-sign-up ${username} <code>' to confirm your account.`
    );
  } catch (err) {
    logger.error(err);
  }
};

export { signUpHandler };

const signUp = ({ clientId, username, password, email }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new SignUpCommand({
    ClientId: clientId,
    Username: username,
    Password: password,
    UserAttributes: [{ Name: "email", Value: email }],
  });

  return client.send(command);
};

import { logger } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { confirmSignUp } from "../../actions/confirm-sign-up.js";
import { FILE_USER_POOLS } from "../constants.js";

```

```
import { getSecondValuesFromEntries } from "@aws-doc-sdk-examples/lib/utils/util-csv.js";

const validateClient = (clientId) => {
  if (!clientId) {
    throw new Error(
      `App client id is missing. Did you run 'create-user-pool'?`,
    );
  }
};

const validateUser = (username) => {
  if (!username) {
    throw new Error(
      `Username name is missing. It must be provided as an argument to the 'confirm-sign-up' command.`,
    );
  }
};

const validateCode = (code) => {
  if (!code) {
    throw new Error(
      `Verification code is missing. It must be provided as an argument to the 'confirm-sign-up' command.`,
    );
  }
};

const confirmSignUpHandler = async (commands) => {
  const [, username, code] = commands;

  try {
    validateUser(username);
    validateCode(code);
    /**
     * @type {string[]}
     */
    const values = getSecondValuesFromEntries(FILE_USER_POOLS);
    const clientId = values[0];
    validateClient(clientId);
    logger.log("Confirming user.");
    await confirmSignUp({ clientId, username, code });
    logger.log(
```

```

        `User confirmed. Run 'admin-initiate-auth ${username} <password>' to sign
        in.` ,
    );
    } catch (err) {
        logger.error(err);
    }
};

export { confirmSignUpHandler };

const confirmSignUp = ({ clientId, username, code }) => {
    const client = new CognitoIdentityProviderClient({});

    const command = new ConfirmSignUpCommand({
        ClientId: clientId,
        Username: username,
        ConfirmationCode: code,
    });

    return client.send(command);
};

import qrcode from "qrcode-terminal";
import { logger } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { adminInitiateAuth } from "../../actions/admin-initiate-auth.js";
import { associateSoftwareToken } from "../../actions/associate-software-token.js";
import { FILE_USER_POOLS } from "../constants.js";
import { getFirstEntry } from "@aws-doc-sdk-examples/lib/utils/util-csv.js";

const handleMfaSetup = async (session, username) => {
    const { SecretCode, Session } = await associateSoftwareToken(session);

    // Store the Session for use with 'VerifySoftwareToken'.
    process.env.SESSION = Session;

    console.log(
        "Scan this code in your preferred authenticator app, then run 'verify-software-token' to finish the setup.",
    );
    qrcode.generate(
        `otpauth://totp/${username}?secret=${SecretCode}`,
        { small: true },
    );
    console.log,

```

```
    );  
  };  
  
  const handleSoftwareTokenMfa = (session) => {  
    // Store the Session for use with 'AdminRespondToAuthChallenge'.  
    process.env.SESSION = session;  
  };  
  
  const validateClient = (id) => {  
    if (!id) {  
      throw new Error(  
        `User pool client id is missing. Did you run 'create-user-pool'?`,  
      );  
    }  
  };  
  
  const validateId = (id) => {  
    if (!id) {  
      throw new Error(`User pool id is missing. Did you run 'create-user-pool'?`);  
    }  
  };  
  
  const validateUser = (username, password) => {  
    if (!(username && password)) {  
      throw new Error(  
        `Username and password must be provided as arguments to the 'admin-initiate-auth' command.`,  
      );  
    }  
  };  
  
  const adminInitiateAuthHandler = async (commands) => {  
    const [, username, password] = commands;  
  
    try {  
      validateUser(username, password);  
  
      const [userPoolId, clientId] = getFirstEntry(FILE_USER_POOLS);  
      validateId(userPoolId);  
      validateClient(clientId);  
  
      logger.log("Signing in.");  
      const { ChallengeName, Session } = await adminInitiateAuth({  
        clientId,
```

```

        userPoolId,
        username,
        password,
    });

    if (ChallengeName === "MFA_SETUP") {
        logger.log("MFA setup is required.");
        return handleMfaSetup(Session, username);
    }

    if (ChallengeName === "SOFTWARE_TOKEN_MFA") {
        handleSoftwareTokenMfa(Session);
        logger.log(`Run 'admin-respond-to-auth-challenge ${username} <totp>'`);
    }
} catch (err) {
    logger.error(err);
}
};

export { adminInitiateAuthHandler };

const adminInitiateAuth = ({ clientId, userPoolId, username, password }) => {
    const client = new CognitoIdentityProviderClient({});

    const command = new AdminInitiateAuthCommand({
        ClientId: clientId,
        UserPoolId: userPoolId,
        AuthFlow: AuthFlowType.ADMIN_USER_PASSWORD_AUTH,
        AuthParameters: { USERNAME: username, PASSWORD: password },
    });

    return client.send(command);
};

import { logger } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { adminRespondToAuthChallenge } from "../../actions/admin-respond-to-auth-challenge.js";
import { getFirstEntry } from "@aws-doc-sdk-examples/lib/utils/util-csv.js";
import { FILE_USER_POOLS } from "../constants.js";

const verifyUsername = (username) => {
    if (!username) {
        throw new Error(

```

```
    `Username is missing. It must be provided as an argument to the 'admin-
respond-to-auth-challenge' command.`,
  );
}
};

const verifyTotp = (totp) => {
  if (!totp) {
    throw new Error(
      `Time-based one-time password (TOTP) is missing. It must be provided as an
argument to the 'admin-respond-to-auth-challenge' command.`,
    );
  }
};

const storeAccessToken = (token) => {
  process.env.AccessToken = token;
};

const adminRespondToAuthChallengeHandler = async (commands) => {
  const [, username, totp] = commands;

  try {
    verifyUsername(username);
    verifyTotp(totp);

    const [userPoolId, clientId] = getFirstEntry(FILE_USER_POOLS);
    const session = process.env.SESSION;

    const { AuthenticationResult } = await adminRespondToAuthChallenge({
      clientId,
      userPoolId,
      username,
      totp,
      session,
    });

    storeAccessToken(AuthenticationResult.AccessToken);

    logger.log("Successfully authenticated.");
  } catch (err) {
    logger.error(err);
  }
};
```

```
export { adminRespondToAuthChallengeHandler };

const respondToAuthChallenge = ({
  clientId,
  username,
  session,
  userPoolId,
  code,
}) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new RespondToAuthChallengeCommand({
    ChallengeName: ChallengeNameType.SOFTWARE_TOKEN_MFA,
    ChallengeResponses: {
      SOFTWARE_TOKEN_MFA_CODE: code,
      USERNAME: username,
    },
    ClientId: clientId,
    UserPoolId: userPoolId,
    Session: session,
  });

  return client.send(command);
};

import { logger } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { verifySoftwareToken } from "../../actions/verify-software-token.js";

const validateTotp = (totp) => {
  if (!totp) {
    throw new Error(
      `Time-based one-time password (TOTP) must be provided to the 'validate-software-token' command.`
    );
  }
};

const verifySoftwareTokenHandler = async (commands) => {
  const [, totp] = commands;

  try {
    validateTotp(totp);

    logger.log("Verifying TOTP.");
  }
};
```



```
    await verifySoftwareToken(totp);
    logger.log("TOTP Verified. Run 'admin-initiate-auth' again to sign-in.");
  } catch (err) {
    logger.error(err);
  }
};

export { verifySoftwareTokenHandler };

const verifySoftwareToken = (totp) => {
  const client = new CognitoIdentityProviderClient({});

  // The 'Session' is provided in the response to 'AssociateSoftwareToken'.
  const session = process.env.SESSION;

  if (!session) {
    throw new Error(
      "Missing a valid Session. Did you run 'admin-initiate-auth'?",
    );
  }

  const command = new VerifySoftwareTokenCommand({
    Session: session,
    UserCode: totp,
  });

  return client.send(command);
};
```

- API-Details finden Sie in den folgenden Themen der AWS SDK für JavaScript -API-Referenz.

- [AdminGetUser](#)
- [AdminInitiateAuth](#)
- [AdminRespondToAuthChallenge](#)
- [AssociateSoftwareToken](#)
- [ConfirmDevice](#)
- [ConfirmSignUp](#)
- [InitiateAuth](#)
- [ListUsers](#)
- [ResendConfirmationCode](#)

- [RespondToAuthChallenge](#)
- [SignUp](#)
- [VerifySoftwareToken](#)

Amazon Comprehend Comprehend-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK für JavaScript (v3) mit Amazon Comprehend Aktionen ausführen und allgemeine Szenarien implementieren.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb eines Service aufrufen oder mit anderen AWS-Services kombinieren.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kodex finden.

Themen

- [Szenarien](#)

Szenarien

Erstellen einer Amazon-Transcribe-Streaming-App

Das folgende Code-Beispiel zeigt, wie Sie eine App erstellen, die Live-Audio in Echtzeit aufzeichnet, transkribiert und übersetzt und die Ergebnisse per E-Mail sendet.

SDK für JavaScript (v3)

Zeigt, wie Amazon Transcribe verwendet wird, um eine App zu erstellen, die Live-Audio in Echtzeit aufzeichnet, transkribiert und übersetzt und die Ergebnisse mit Amazon Simple Email Service (Amazon SES) per E-Mail sendet.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

In diesem Beispiel verwendete Dienste

- Amazon Comprehend

- Amazon SES
- Amazon Transcribe
- Amazon Translate

Erstellen eines Amazon-Lex-Chatbots

Das folgende Codebeispiele zeigt, wie Sie einen Chatbot erstellen können, um die Besucher Ihrer Website einzubinden.

SDK für JavaScript (v3)

Zeigt, wie Sie die Amazon-Lex-API verwenden können, um einen Chatbot innerhalb einer Webanwendung zu erstellen, um die Besucher Ihrer Website einzubinden.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel „[Einen Amazon Lex-Chatbot erstellen](#)“ im AWS SDK für JavaScript Entwicklerhandbuch.

In diesem Beispiel verwendete Dienste

- Amazon Comprehend
- Amazon Lex
- Amazon Translate

Erstellen einer Anwendung zum Analysieren von Kundenfeedback

Das folgende Codebeispiel zeigt, wie Sie eine Anwendung erstellen, die Kundenkommentarkarten analysiert, sie aus der ursprünglichen Sprache übersetzt, die Stimmung ermittelt und auf der Grundlage des übersetzten Texts eine Audiodatei generiert.

SDK für JavaScript (v3)

Diese Beispielanwendung analysiert und speichert Kundenfeedback-Karten. Sie ist auf die Anforderungen eines fiktiven Hotels in New York City zugeschnitten. Das Hotel erhält Feedback von Gästen in Form von physischen Kommentarkarten in verschiedenen Sprachen. Dieses Feedback wird über einen Webclient in die App hochgeladen. Nachdem ein Bild einer Kommentarkarte hochgeladen wurde, werden folgende Schritte ausgeführt:

- Der Text wird mithilfe von Amazon Textract aus dem Bild extrahiert.
- Amazon Comprehend ermittelt die Stimmung und die Sprache des extrahierten Textes.

- Der extrahierte Text wird mithilfe von Amazon Translate ins Englische übersetzt.
- Amazon Polly generiert auf der Grundlage des extrahierten Texts eine Audiodatei.

Die vollständige App kann mithilfe des AWS CDK bereitgestellt werden. Quellcode und Anweisungen zur Bereitstellung finden Sie im Projekt unter [GitHub](#). Die folgenden Auszüge zeigen, wie der innerhalb von Lambda-Funktionen verwendet AWS SDK für JavaScript wird.

```
import {
  ComprehendClient,
  DetectDominantLanguageCommand,
  DetectSentimentCommand,
} from "@aws-sdk/client-comprehend";

/**
 * Determine the language and sentiment of the extracted text.
 *
 * @param {{ source_text: string }} extractTextOutput
 */
export const handler = async (extractTextOutput) => {
  const comprehendClient = new ComprehendClient({});

  const detectDominantLanguageCommand = new DetectDominantLanguageCommand({
    Text: extractTextOutput.source_text,
  });

  // The source language is required for sentiment analysis and
  // translation in the next step.
  const { Languages } = await comprehendClient.send(
    detectDominantLanguageCommand,
  );

  const languageCode = Languages[0].LanguageCode;

  const detectSentimentCommand = new DetectSentimentCommand({
    Text: extractTextOutput.source_text,
    LanguageCode: languageCode,
  });

  const { Sentiment } = await comprehendClient.send(detectSentimentCommand);

  return {
    sentiment: Sentiment,
    language_code: languageCode,
  };
};
```

```
};
};
```

```
import {
  DetectDocumentTextCommand,
  TextractClient,
} from "@aws-sdk/client-textract";

/**
 * Fetch the S3 object from the event and analyze it using Amazon Textract.
 *
 * @param {import("@types/aws-lambda").EventBridgeEvent<"Object Created">}
  eventBridgeS3Event
 */
export const handler = async (eventBridgeS3Event) => {
  const textractClient = new TextractClient();

  const detectDocumentTextCommand = new DetectDocumentTextCommand({
    Document: {
      S3Object: {
        Bucket: eventBridgeS3Event.bucket,
        Name: eventBridgeS3Event.object,
      },
    },
  });

  // Textract returns a list of blocks. A block can be a line, a page, word, etc.
  // Each block also contains geometry of the detected text.
  // For more information on the Block type, see https://docs.aws.amazon.com/textract/latest/dg/API\_Block.html.
  const { Blocks } = await textractClient.send(detectDocumentTextCommand);

  // For the purpose of this example, we are only interested in words.
  const extractedWords = Blocks.filter((b) => b.BlockType === "WORD").map(
    (b) => b.Text,
  );

  return extractedWords.join(" ");
};
```

```
import { PollyClient, SynthesizeSpeechCommand } from "@aws-sdk/client-polly";
import { S3Client } from "@aws-sdk/client-s3";
import { Upload } from "@aws-sdk/lib-storage";
```

```

/**
 * Synthesize an audio file from text.
 *
 * @param {{ bucket: string, translated_text: string, object: string }}
 * sourceDestinationConfig
 */
export const handler = async (sourceDestinationConfig) => {
  const pollyClient = new PollyClient({});

  const synthesizeSpeechCommand = new SynthesizeSpeechCommand({
    Engine: "neural",
    Text: sourceDestinationConfig.translated_text,
    VoiceId: "Ruth",
    OutputFormat: "mp3",
  });

  const { AudioStream } = await pollyClient.send(synthesizeSpeechCommand);

  const audioKey = `${sourceDestinationConfig.object}.mp3`;

  // Store the audio file in S3.
  const s3Client = new S3Client();
  const upload = new Upload({
    client: s3Client,
    params: {
      Bucket: sourceDestinationConfig.bucket,
      Key: audioKey,
      Body: AudioStream,
      ContentType: "audio/mp3",
    },
  });

  await upload.done();
  return audioKey;
};

```

```

import {
  TranslateClient,
  TranslateTextCommand,
} from "@aws-sdk/client-translate";

/**

```

```
* Translate the extracted text to English.
*
* @param {{ extracted_text: string, source_language_code: string }}
textAndSourceLanguage
*/
export const handler = async (textAndSourceLanguage) => {
  const translateClient = new TranslateClient({});

  const translateCommand = new TranslateTextCommand({
    SourceLanguageCode: textAndSourceLanguage.source_language_code,
    TargetLanguageCode: "en",
    Text: textAndSourceLanguage.extracted_text,
  });

  const { TranslatedText } = await translateClient.send(translateCommand);

  return { translated_text: TranslatedText };
};
```

In diesem Beispiel verwendete Dienste

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

Amazon DocumentDB DocumentDB-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie AWS SDK für JavaScript (v3) mit Amazon DocumentDB verwenden.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kodex finden.

Themen

- [Serverless-Beispiele](#)

Serverless-Beispiele

Aufrufen einer Lambda-Funktion über einen Amazon DocumentDB-Auslöser

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein durch den Empfang von Datensätzen aus einem DocumentDB-Änderungsstream ausgelöstes Ereignis empfängt. Die Funktion ruft die DocumentDB-Nutzdaten ab und protokolliert den Inhalt des Datensatzes.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Verwenden eines Amazon DocumentDB DocumentDB-Ereignisses mit Lambda. JavaScript

```
console.log('Loading function');
exports.handler = async (event, context) => {
  event.events.forEach(record => {
    logDocumentDBEvent(record);
  });
  return 'OK';
};

const logDocumentDBEvent = (record) => {
  console.log('Operation type: ' + record.event.operationType);
  console.log('db: ' + record.event.ns.db);
  console.log('collection: ' + record.event.ns.coll);
  console.log('Full document:', JSON.stringify(record.event.fullDocument, null,
    2));
};
```

Ein Amazon DocumentDB DocumentDB-Ereignis mit Lambda verwenden TypeScript

```
import { DocumentDBEventRecord, DocumentDBEventSubscriptionContext } from 'aws-
lambda';
```



```
console.log('Loading function');

export const handler = async (
  event: DocumentDBEventSubscriptionContext,
  context: any
): Promise<string> => {
  event.events.forEach((record: DocumentDBEventRecord) => {
    logDocumentDBEvent(record);
  });
  return 'OK';
};

const logDocumentDBEvent = (record: DocumentDBEventRecord): void => {
  console.log('Operation type: ' + record.event.operationType);
  console.log('db: ' + record.event.ns.db);
  console.log('collection: ' + record.event.ns.coll);
  console.log('Full document:', JSON.stringify(record.event.fullDocument, null, 2));
};
```

DynamoDB-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie AWS SDK für JavaScript (v3) mit DynamoDB verwenden.

Bei Grundlagen handelt es sich um Codebeispiele, die Ihnen zeigen, wie Sie die wesentlichen Vorgänge innerhalb eines Services ausführen.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien anzeigen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb eines Service aufrufen oder mit anderen AWS-Services kombinieren.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kodex finden.

Themen

- [Erste Schritte](#)

- [Grundlagen](#)
- [Aktionen](#)
- [Szenarien](#)
- [Serverless-Beispiele](#)

Erste Schritte

Hello DynamoDB

Das folgenden Codebeispiel veranschaulicht die ersten Schritte mit DynamoDB.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Weitere Informationen zur Arbeit mit DynamoDB finden Sie unter [DynamoDB programmieren](#) mit AWS SDK für JavaScript JavaScript

```
import { ListTablesCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new ListTablesCommand({});

  const response = await client.send(command);
  console.log(response.TableNames.join("\n"));
  return response;
};
```

- Einzelheiten zur API finden Sie unter [ListTables](#)API-Referenz.AWS SDK für JavaScript

Grundlagen

Kennenlernen der Grundlagen

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Erstellen einer Tabelle, die Filmdaten enthalten kann.
- Einfügen, Abrufen und Aktualisieren eines einzelnen Films in der Tabelle.
- Schreiben von Filmdaten in die Tabelle anhand einer JSON-Beispieldatei.
- Abfragen nach Filmen, die in einem bestimmten Jahr veröffentlicht wurden.
- Scan nach Filmen, die in mehreren Jahren veröffentlicht wurden.
- Löschen eines Films aus der Tabelle und anschließendes Löschen der Tabelle.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { readFileSync } from "node:fs";
import {
  BillingMode,
  CreateTableCommand,
  DeleteTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";

/**
 * This module is a convenience library. It abstracts Amazon DynamoDB's data type
 * descriptors (such as S, N, B, and B00L) by marshalling JavaScript objects into
 * AttributeValue shapes.
 */
import {
  BatchWriteCommand,
  DeleteCommand,
  DynamoDBDocumentClient,
```

```

    GetCommand,
    PutCommand,
    UpdateCommand,
    paginateQuery,
    paginateScan,
  } from "@aws-sdk/lib-dynamodb";

// These modules are local to our GitHub repository. We recommend cloning
// the project from GitHub if you want to run this example.
// For more information, see https://github.com/awsdocs/aws-doc-sdk-examples.
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import { chunkArray } from "@aws-doc-sdk-examples/lib/utils/util-array.js";

const dirname = dirnameFromMetaUrl(import.meta.url);
const tableName = getUniqueName("Movies");
const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

const log = (msg) => console.log(`[SCENARIO] ${msg}`);

export const main = async () => {
  /**
   * Create a table.
   */

  const createTableCommand = new CreateTableCommand({
    TableName: tableName,
    // This example performs a large write to the database.
    // Set the billing mode to PAY_PER_REQUEST to
    // avoid throttling the large write.
    BillingMode: BillingMode.PAY_PER_REQUEST,
    // Define the attributes that are necessary for the key schema.
    AttributeDefinitions: [
      {
        AttributeName: "year",
        // 'N' is a data type descriptor that represents a number type.
        // For a list of all data type descriptors, see the following link.
        // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
        AttributeType: "N",
      },
      { AttributeName: "title", AttributeType: "S" },
    ],
  },

```

```

// The KeySchema defines the primary key. The primary key can be
// a partition key, or a combination of a partition key and a sort key.
// Key schema design is important. For more info, see
// https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/best-
practices.html
KeySchema: [
  // The way your data is accessed determines how you structure your keys.
  // The movies table will be queried for movies by year. It makes sense
  // to make year our partition (HASH) key.
  { AttributeName: "year", KeyType: "HASH" },
  { AttributeName: "title", KeyType: "RANGE" },
],
});

log("Creating a table.");
const createTableResponse = await client.send(createTableCommand);
log(`Table created: ${JSON.stringify(createTableResponse.TableDescription)}`);

// This polls with DescribeTableCommand until the requested table is 'ACTIVE'.
// You can't write to a table before it's active.
log("Waiting for the table to be active.");
await waitUntilTableExists({ client }, { TableName: tableName });
log("Table active.");

/**
 * Add a movie to the table.
 */

log("Adding a single movie to the table.");
// PutCommand is the first example usage of 'lib-dynamodb'.
const putCommand = new PutCommand({
  TableName: tableName,
  Item: {
    // In 'client-dynamodb', the AttributeValue would be required ( `year: { N:
1981 }` )
    // 'lib-dynamodb' simplifies the usage ( `year: 1981` )
    year: 1981,
    // The preceding KeySchema defines 'title' as our sort (RANGE) key, so 'title'
    // is required.
    title: "The Evil Dead",
    // Every other attribute is optional.
    info: {
      genres: ["Horror"],
    },
  },

```

```

    },
  });
  await docClient.send(putCommand);
  log("The movie was added.");

  /**
   * Get a movie from the table.
   */

  log("Getting a single movie from the table.");
  const getCommand = new GetCommand({
    TableName: tableName,
    // Requires the complete primary key. For the movies table, the primary key
    // is only the id (partition key).
    Key: {
      year: 1981,
      title: "The Evil Dead",
    },
    // Set this to make sure that recent writes are reflected.
    // For more information, see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.ReadConsistency.html.
    ConsistentRead: true,
  });
  const getResponse = await docClient.send(getCommand);
  log(`Got the movie: ${JSON.stringify(getResponse.Item)}`);

  /**
   * Update a movie in the table.
   */

  log("Updating a single movie in the table.");
  const updateCommand = new UpdateCommand({
    TableName: tableName,
    Key: { year: 1981, title: "The Evil Dead" },
    // This update expression appends "Comedy" to the list of genres.
    // For more information on update expressions, see
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Expressions.UpdateExpressions.html
    UpdateExpression: "set #i.#g = list_append(#i.#g, :vals)",
    ExpressionAttributeNames: { "#i": "info", "#g": "genres" },
    ExpressionAttributeValues: {
      ":vals": ["Comedy"],
    },
    ReturnValues: "ALL_NEW",
  });

```

```
});
const updateResponse = await docClient.send(updateCommand);
log(`Movie updated: ${JSON.stringify(updateResponse.Attributes)}`);

/**
 * Delete a movie from the table.
 */

log("Deleting a single movie from the table.");
const deleteCommand = new DeleteCommand({
  TableName: tableName,
  Key: { year: 1981, title: "The Evil Dead" },
});
await docClient.send(deleteCommand);
log("Movie deleted.");

/**
 * Upload a batch of movies.
 */

log("Adding movies from local JSON file.");
const file = readFileSync(
  `${dirname}../../../../resources/sample_files/movies.json`,
);
const movies = JSON.parse(file.toString());
// chunkArray is a local convenience function. It takes an array and returns
// a generator function. The generator function yields every N items.
const movieChunks = chunkArray(movies, 25);
// For every chunk of 25 movies, make one BatchWrite request.
for (const chunk of movieChunks) {
  const putRequests = chunk.map((movie) => ({
    PutRequest: {
      Item: movie,
    },
  }));

  const command = new BatchWriteCommand({
    RequestItems: {
      [tableName]: putRequests,
    },
  });

  await docClient.send(command);
}
```

```

log("Movies added.");

/**
 * Query for movies by year.
 */

log("Querying for all movies from 1981.");
const paginatedQuery = paginateQuery(
  { client: docClient },
  {
    TableName: tableName,
    //For more information about query expressions, see
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
    Query.html#Query.KeyConditionExpressions
    KeyConditionExpression: "#y = :y",
    // 'year' is a reserved word in DynamoDB. Indicate that it's an attribute
    // name by using an expression attribute name.
    ExpressionAttributeNames: { "#y": "year" },
    ExpressionAttributeValues: { ":y": 1981 },
    ConsistentRead: true,
  },
);
/**
 * @type { Record<string, any>[] };
 */
const movies1981 = [];
for await (const page of paginatedQuery) {
  movies1981.push(...page.Items);
}
log(`Movies: ${movies1981.map((m) => m.title).join(", ")}`);

/**
 * Scan the table for movies between 1980 and 1990.
 */

log("Scan for movies released between 1980 and 1990");
// A 'Scan' operation always reads every item in the table. If your design
requires
// the use of 'Scan', consider indexing your table or changing your design.
// https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/bp-query-
scan.html
const paginatedScan = paginateScan(
  { client: docClient },
  {

```



```

    TableName: tableName,
    // Scan uses a filter expression instead of a key condition expression. Scan
will
    // read the entire table and then apply the filter.
    FilterExpression: "#y between :y1 and :y2",
    ExpressionAttributeNames: { "#y": "year" },
    ExpressionAttributeValues: { ":y1": 1980, ":y2": 1990 },
    ConsistentRead: true,
  },
);
/**
 * @type { Record<string, any>[] };
 */
const movies1980to1990 = [];
for await (const page of paginatedScan) {
  movies1980to1990.push(...page.Items);
}
log(
  `Movies: ${movies1980to1990
    .map((m) => `${m.title} (${m.year})`)
    .join(", ")}`,
);

/**
 * Delete the table.
 */

const deleteTableCommand = new DeleteTableCommand({ TableName: tableName });
log(`Deleting table ${tableName}.`);
await client.send(deleteTableCommand);
log("Table deleted.");
};

```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für JavaScript - API-Referenz.
 - [BatchWriteItem](#)
 - [CreateTable](#)
 - [DeleteItem](#)
 - [DeleteTable](#)
 - [DescribeTable](#)

- [GetItem](#)
- [PutItem](#)
- [Query](#)
- [Scan](#)
- [UpdateItem](#)

Aktionen

BatchExecuteStatement

Das folgende Codebeispiel zeigt die Verwendung `BatchExecuteStatement`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie mithilfe von PartiQL einen Stapel von Elementen.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const breakfastFoods = ["Eggs", "Bacon", "Sausage"];
  const command = new BatchExecuteStatementCommand({
    Statements: breakfastFoods.map((food) => ({
      Statement: `INSERT INTO BreakfastFoods value {'Name':?}`,
      Parameters: [food],
    })),
  });
```

```
});

const response = await docClient.send(command);
console.log(response);
return response;
};
```

Rufen Sie mithilfe von PartiQL einen Stapel von Elementen ab.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new BatchExecuteStatementCommand({
    Statements: [
      {
        Statement: "SELECT * FROM PepperMeasurements WHERE Unit=?",
        Parameters: ["Teaspoons"],
        ConsistentRead: true,
      },
      {
        Statement: "SELECT * FROM PepperMeasurements WHERE Unit=?",
        Parameters: ["Grams"],
        ConsistentRead: true,
      },
    ],
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

Aktualisieren Sie mithilfe von PartiQL einen Stapel von Elementen.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const eggUpdates = [
    ["duck", "fried"],
    ["chicken", "omelette"],
  ];
  const command = new BatchExecuteStatementCommand({
    Statements: eggUpdates.map((change) => ({
      Statement: "UPDATE Eggs SET Style=? where Variety=?",
      Parameters: [change[1], change[0]],
    })),
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

Löschen Sie mithilfe von PartiQL einen Stapel von Elementen.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new BatchExecuteStatementCommand({
    Statements: [
```

```
{
  Statement: "DELETE FROM Flavors where Name=?",
  Parameters: ["Grape"],
},
{
  Statement: "DELETE FROM Flavors where Name=?",
  Parameters: ["Strawberry"],
},
],
});

const response = await docClient.send(command);
console.log(response);
return response;
};
```

- Einzelheiten zur API finden Sie [BatchExecuteStatement](#) in der AWS SDK für JavaScript API-Referenz.

BatchGetItem

Das folgende Codebeispiel zeigt die Verwendung `BatchGetItem`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

In diesem Beispiel wird der Dokument-Client verwendet, um die Arbeit mit Elementen in DynamoDB zu vereinfachen. Einzelheiten zur API finden Sie unter [BatchGet](#).

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { BatchGetCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);
```

```
export const main = async () => {
  const command = new BatchGetCommand({
    // Each key in this object is the name of a table. This example refers
    // to a Books table.
    RequestItems: {
      Books: {
        // Each entry in Keys is an object that specifies a primary key.
        Keys: [
          {
            Title: "How to AWS",
          },
          {
            Title: "DynamoDB for DBAs",
          },
        ],
        // Only return the "Title" and "PageCount" attributes.
        ProjectionExpression: "Title, PageCount",
      },
    },
  });

  const response = await docClient.send(command);
  console.log(response.Responses.Books);
  return response;
};
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [BatchGetItem](#) in der AWS SDK für JavaScript API-Referenz.

BatchWriteItem

Das folgende Codebeispiel zeigt die Verwendung `BatchWriteItem`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

In diesem Beispiel wird der Dokument-Client verwendet, um die Arbeit mit Elementen in DynamoDB zu vereinfachen. Einzelheiten zur API finden Sie unter [BatchWrite](#).

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  BatchWriteCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";
import { readFileSync } from "node:fs";

// These modules are local to our GitHub repository. We recommend cloning
// the project from GitHub if you want to run this example.
// For more information, see https://github.com/awsdocs/aws-doc-sdk-examples.
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import { chunkArray } from "@aws-doc-sdk-examples/lib/utils/util-array.js";

const dirname = dirnameFromMetaUrl(import.meta.url);

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const file = readFileSync(
    `${dirname}../../../../resources/sample_files/movies.json`,
  );

  const movies = JSON.parse(file.toString());

  // chunkArray is a local convenience function. It takes an array and returns
  // a generator function. The generator function yields every N items.
  const movieChunks = chunkArray(movies, 25);

  // For every chunk of 25 movies, make one BatchWrite request.
  for (const chunk of movieChunks) {
    const putRequests = chunk.map((movie) => ({
      PutRequest: {
        Item: movie,
      },
    }));

    const command = new BatchWriteCommand({
      RequestItems: {
```

```

        // An existing table is required. A composite key of 'title' and 'year' is
        recommended
        // to account for duplicate titles.
        BatchWriteMoviesTable: putRequests,
    },
});

await docClient.send(command);
}
};

```

- Einzelheiten zur API finden Sie [BatchWriteItem](#) in der AWS SDK für JavaScript API-Referenz.

CreateTable

Das folgende Codebeispiel zeigt die Verwendung `CreateTable`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```

import { CreateTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
    const command = new CreateTableCommand({
        TableName: "EspressoDrinks",
        // For more information about data types,
        // see https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
        // HowItWorks.NamingRulesDataTypes.html#HowItWorks.DataTypes and
        // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
        // Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
        AttributeDefinitions: [
            {
                AttributeName: "DrinkName",
                AttributeType: "S",
            },
        ],
    });
};

```



```
    },
  ],
  KeySchema: [
    {
      AttributeName: "DrinkName",
      KeyType: "HASH",
    },
  ],
  BillingMode: "PAY_PER_REQUEST",
});

const response = await client.send(command);
console.log(response);
return response;
};
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [CreateTable](#) in der AWS SDK für JavaScript API-Referenz.

DeleteItem

Das folgende Codebeispiel zeigt die Verwendung `DeleteItem`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel](#) einrichten und ausführen.

In diesem Beispiel wird der Dokument-Client verwendet, um die Arbeit mit Elementen in DynamoDB zu vereinfachen. Einzelheiten zur API finden Sie unter [DeleteCommand](#).

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, DeleteCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);
```

```
export const main = async () => {
  const command = new DeleteCommand({
    TableName: "Sodas",
    Key: {
      Flavor: "Cola",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [DeleteItem](#) in der AWS SDK für JavaScript API-Referenz.

DeleteTable

Das folgende Codebeispiel zeigt die Verwendung `DeleteTable`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { DeleteTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new DeleteTableCommand({
    TableName: "DecafCoffees",
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

```
};
```

- Einzelheiten zur API finden Sie [DeleteTable](#) in der AWS SDK für JavaScript API-Referenz.

DescribeTable

Das folgende Codebeispiel zeigt die Verwendung `DescribeTable`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { DescribeTableCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new DescribeTableCommand({
    TableName: "Pastries",
  });

  const response = await client.send(command);
  console.log(`TABLE NAME: ${response.Table.TableName}`);
  console.log(`TABLE ITEM COUNT: ${response.Table.ItemCount}`);
  return response;
};
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [DescribeTable](#) in der AWS SDK für JavaScript API-Referenz.

DescribeTimeToLive

Das folgende Codebeispiel zeigt die Verwendung `DescribeTimeToLive`.

SDK für JavaScript (v3)

Beschreiben Sie die TTL-Konfiguration für eine bestehende DynamoDB-Tabelle mithilfe von AWS SDK für JavaScript.

```
import { DynamoDBClient, DescribeTimeToLiveCommand } from "@aws-sdk/client-dynamodb";

export const describeTTL = async (tableName, region) => {
  const client = new DynamoDBClient({
    region: region,
    endpoint: `https://dynamodb.${region}.amazonaws.com`
  });

  try {
    const ttlDescription = await client.send(new
DescribeTimeToLiveCommand({ TableName: tableName }));

    if (ttlDescription.TimeToLiveDescription.TimeToLiveStatus === 'ENABLED') {
      console.log("TTL is enabled for table %s.", tableName);
    } else {
      console.log("TTL is not enabled for table %s.", tableName);
    }

    return ttlDescription;
  } catch (e) {
    console.error(`Error describing table: ${e}`);
    throw e;
  }
}

// Example usage (commented out for testing)
// describeTTL('your-table-name', 'us-east-1');
```

- Einzelheiten zur API finden Sie [DescribeTimeToLive](#) in der AWS SDK für JavaScript API-Referenz.

ExecuteStatement

Das folgende Codebeispiel zeigt die Verwendung `ExecuteStatement`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie ein Element mithilfe von PartiQL.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  ExecuteStatementCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: `INSERT INTO Flowers value {'Name':?}`,
    Parameters: ["Rose"],
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

Rufen Sie ein Element mithilfe von PartiQL ab.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  ExecuteStatementCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
```

```
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: "SELECT * FROM CloudTypes WHERE IsStorm=?",
    Parameters: [false],
    ConsistentRead: true,
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

Aktualisieren Sie ein Element mithilfe von PartiQL.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

import {
  ExecuteStatementCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: "UPDATE EyeColors SET IsRecessive=? where Color=?",
    Parameters: [true, "blue"],
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

Löschen Sie ein Element mithilfe von PartiQL.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
```

```
import {
  ExecuteStatementCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ExecuteStatementCommand({
    Statement: "DELETE FROM PaintColors where Name=?",
    Parameters: ["Purple"],
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Einzelheiten zur API finden Sie [ExecuteStatement](#) in der AWS SDK für JavaScript API-Referenz.

GetItem

Das folgende Codebeispiel zeigt die Verwendung `GetItem`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

In diesem Beispiel wird der Dokument-Client verwendet, um die Arbeit mit Elementen in DynamoDB zu vereinfachen. Einzelheiten zur API finden Sie unter [GetCommand](#).

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, GetCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
```

```
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new GetCommand({
    TableName: "AngryAnimals",
    Key: {
      CommonName: "Shoebill",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Einzelheiten zur API finden Sie [GetItem](#) in der AWS SDK für JavaScript API-Referenz.

ListTables

Das folgende Codebeispiel zeigt die Verwendung `ListTables`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { ListTablesCommand, DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({});

export const main = async () => {
  const command = new ListTablesCommand({});

  const response = await client.send(command);
  console.log(response);
  return response;
};
```


- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [ListTables](#) in der AWS SDK für JavaScript API-Referenz.

PutItem

Das folgende Codebeispiel zeigt die Verwendung `PutItem`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel](#) einrichten und ausführen.

In diesem Beispiel wird der Dokument-Client verwendet, um die Arbeit mit Elementen in DynamoDB zu vereinfachen. Einzelheiten zur API finden Sie unter [PutCommand](#).

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { PutCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new PutCommand({
    TableName: "HappyAnimals",
    Item: {
      CommonName: "Shiba Inu",
    },
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Einzelheiten zur API finden Sie [PutItem](#) in der AWS SDK für JavaScript API-Referenz.

Query

Das folgende Codebeispiel zeigt die Verwendung `Query`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

In diesem Beispiel wird der Dokument-Client verwendet, um die Arbeit mit Elementen in DynamoDB zu vereinfachen. Einzelheiten zur API finden Sie unter [QueryCommand](#).

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { QueryCommand, DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new QueryCommand({
    TableName: "CoffeeCrop",
    KeyConditionExpression:
      "OriginCountry = :originCountry AND RoastDate > :roastDate",
    ExpressionAttributeValues: {
      ":originCountry": "Ethiopia",
      ":roastDate": "2023-05-01",
    },
    ConsistentRead: true,
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Weitere API-Informationen finden Sie unter [Abfragen](#) in der AWS SDK für JavaScript -API-Referenz.

Scan

Das folgende Codebeispiel zeigt die Verwendung `Scan`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

In diesem Beispiel wird der Dokument-Client verwendet, um die Arbeit mit Elementen in DynamoDB zu vereinfachen. Einzelheiten zur API finden Sie unter [ScanCommand](#).

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, ScanCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new ScanCommand({
    ProjectionExpression: "#Name, Color, AvgLifeSpan",
    ExpressionAttributeNames: { "#Name": "Name" },
    TableName: "Birds",
  });

  const response = await docClient.send(command);
  for (const bird of response.Items) {
    console.log(`${bird.Name} - (${bird.Color}, ${bird.AvgLifeSpan})`);
  }
  return response;
};
```

- Weitere API-Informationen finden Sie unter [Scan](#) in der AWS SDK für JavaScript -API-Referenz.

UpdateItem

Das folgende Codebeispiel zeigt die Verwendung `UpdateItem`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

In diesem Beispiel wird der Dokument-Client verwendet, um die Arbeit mit Elementen in DynamoDB zu vereinfachen. Einzelheiten zur API finden Sie unter [UpdateCommand](#).

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, UpdateCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const main = async () => {
  const command = new UpdateCommand({
    TableName: "Dogs",
    Key: {
      Breed: "Labrador",
    },
    UpdateExpression: "set Color = :color",
    ExpressionAttributeValues: {
      ":color": "black",
    },
    ReturnValues: "ALL_NEW",
  });

  const response = await docClient.send(command);
  console.log(response);
  return response;
};
```

- Einzelheiten zur API finden Sie [UpdateItem](#) in der AWS SDK für JavaScript API-Referenz.

UpdateTimeToLive

Das folgende Codebeispiel zeigt die Verwendung `UpdateTimeToLive`.

SDK für JavaScript (v3)

Aktivieren Sie TTL für eine vorhandene DynamoDB-Tabelle.

```
import { DynamoDBClient, UpdateTimeToLiveCommand } from "@aws-sdk/client-dynamodb";

export const enableTTL = async (tableName, ttlAttribute, region = 'us-east-1') => {

  const client = new DynamoDBClient({
    region: region,
    endpoint: `https://dynamodb.${region}.amazonaws.com`
  });

  const params = {
    TableName: tableName,
    TimeToLiveSpecification: {
      Enabled: true,
      AttributeName: ttlAttribute
    }
  };

  try {
    const response = await client.send(new UpdateTimeToLiveCommand(params));
    if (response.$metadata.httpStatusCode === 200) {
      console.log(`TTL enabled successfully for table ${tableName}, using
attribute name ${ttlAttribute}.`);
    } else {
      console.log(`Failed to enable TTL for table ${tableName}, response
object: ${response}`);
    }
    return response;
  } catch (e) {
    console.error(`Error enabling TTL: ${e}`);
    throw e;
  }
};

// Example usage (commented out for testing)
// enableTTL('ExampleTable', 'exampleTtlAttribute');
```

Deaktivieren Sie TTL für eine vorhandene DynamoDB-Tabelle.

```
import { DynamoDBClient, UpdateTimeToLiveCommand } from "@aws-sdk/client-dynamodb";

export const disableTTL = async (tableName, ttlAttribute, region = 'us-east-1') => {

  const client = new DynamoDBClient({
    region: region,
    endpoint: `https://dynamodb.${region}.amazonaws.com`
  });

  const params = {
    TableName: tableName,
    TimeToLiveSpecification: {
      Enabled: false,
      AttributeName: ttlAttribute
    }
  };

  try {
    const response = await client.send(new UpdateTimeToLiveCommand(params));
    if (response.$metadata.httpStatusCode === 200) {
      console.log(`TTL disabled successfully for table ${tableName}, using
attribute name ${ttlAttribute}.`);
    } else {
      console.log(`Failed to disable TTL for table ${tableName}, response
object: ${response}`);
    }
    return response;
  } catch (e) {
    console.error(`Error disabling TTL: ${e}`);
    throw e;
  }
};

// Example usage (commented out for testing)
// disableTTL('ExampleTable', 'exampleTtlAttribute');
```

- Einzelheiten zur API finden Sie [UpdateTimeToLive](#) in der AWS SDK für JavaScript API-Referenz.

Szenarien

Erstellen Sie eine App zum Senden von Daten an eine DynamoDB-Tabelle

Die folgenden Codebeispiele zeigen, wie eine Anwendung erstellt wird, die Daten an eine Amazon-DynamoDB-Tabelle übermittelt und Sie benachrichtigt, wenn ein Benutzer die Tabelle aktualisiert.

SDK für JavaScript (v3)

Das Beispiel zeigt, wie man eine App erstellt, die es Benutzern ermöglicht, Daten an eine Amazon-DynamoDB-Tabelle zu übermitteln und eine Textnachricht an den Administrator mit Amazon Simple Notification Service (Amazon SNS) zu senden.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

Dieses Beispiel ist auch verfügbar im [AWS SDK für JavaScript Entwicklerhandbuch für v3](#).

In diesem Beispiel verwendete Dienste

- DynamoDB
- Amazon SNS

Vergleichen mehrerer Werte mit einem einzigen Attribut

Im folgenden Codebeispiel wird gezeigt, wie Sie mehrere Werte mit einem einzigen Attribut in DynamoDB vergleichen.

- Verwenden des IN-Operators, um mehrere Werte mit einem einzigen Attribut zu vergleichen
- Vergleichen des IN-Operators mit mehreren OR-Bedingungen
- Verständnis der Vorteilen von IN hinsichtlich Leistung und Ausdruckskomplexität

SDK für JavaScript (v3)

Vergleichen Sie mehrere Werte mit einem einzigen Attribut mithilfe von AWS SDK für JavaScript.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  ScanCommand,
  QueryCommand
}
```

```

} = require("@aws-sdk/lib-dynamodb");

/**
 * Query or scan a DynamoDB table to find items where an attribute matches any value
 * from a list.
 *
 * This function demonstrates the use of the IN operator to compare a single
 * attribute
 * against multiple possible values, which is more efficient than using multiple OR
 * conditions.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} attributeName - The name of the attribute to compare against the
 * values list
 * @param {Array} valuesList - List of values to compare the attribute against
 * @param {string} [partitionKeyName] - Optional name of the partition key attribute
 * for query operations
 * @param {string} [partitionKeyValue] - Optional value of the partition key to
 * query
 * @returns {Promise<Object>} - The response from DynamoDB containing the matching
 * items
 */
async function compareMultipleValues(
  config,
  tableName,
  attributeName,
  valuesList,
  partitionKeyName,
  partitionKeyValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Create the filter expression using the IN operator
  const filterExpression = `${attributeName} IN (${valuesList.map((_, index) =>
    `:val${index}`}).join(', ')});`;

  // Create expression attribute values for the values list
  const expressionAttributeValues = valuesList.reduce((acc, val, index) => {
    acc[`val${index}`] = val;
    return acc;
  }, {});

```



```
// If partition key is provided, perform a query operation
if (partitionKeyName && partitionKeyValue) {
  const keyCondition = `${partitionKeyName} = :partitionKey`;
  expressionAttributeValues[':partitionKey'] = partitionKeyValue;

  // Initialize array to collect all items
  let allItems = [];
  let lastEvaluatedKey;

  // Use pagination to get all results
  do {
    const params = {
      TableName: tableName,
      KeyConditionExpression: keyCondition,
      FilterExpression: filterExpression,
      ExpressionAttributeValues: expressionAttributeValues
    };

    // Add ExclusiveStartKey if we have a lastEvaluatedKey from a previous query
    if (lastEvaluatedKey) {
      params.ExclusiveStartKey = lastEvaluatedKey;
    }

    const response = await docClient.send(new QueryCommand(params));

    // Add the items from this page to our collection
    if (response.Items && response.Items.length > 0) {
      allItems = [...allItems, ...response.Items];
    }

    // Get the key for the next page of results
    lastEvaluatedKey = response.LastEvaluatedKey;
  } while (lastEvaluatedKey);

  // Return the complete result
  return {
    Items: allItems,
    Count: allItems.length
  };
} else {
  // Otherwise, perform a scan operation
  // Initialize array to collect all items
  let allItems = [];
```

```
let lastEvaluatedKey;

// Use pagination to get all results
do {
  const params = {
    TableName: tableName,
    FilterExpression: filterExpression,
    ExpressionAttributeValues: expressionAttributeValues
  };

  // Add ExclusiveStartKey if we have a lastEvaluatedKey from a previous scan
  if (lastEvaluatedKey) {
    params.ExclusiveStartKey = lastEvaluatedKey;
  }

  const response = await docClient.send(new ScanCommand(params));

  // Add the items from this page to our collection
  if (response.Items && response.Items.length > 0) {
    allItems = [...allItems, ...response.Items];
  }

  // Get the key for the next page of results
  lastEvaluatedKey = response.LastEvaluatedKey;
} while (lastEvaluatedKey);

// Return the complete result
return {
  Items: allItems,
  Count: allItems.length
};
}

/**
 * Alternative implementation using multiple OR conditions instead of the IN
 * operator.
 *
 * This function is provided for comparison to show why using the IN operator is
 * preferable.
 * With many values, this approach becomes verbose and less efficient.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table

```

```

* @param {string} attributeName - The name of the attribute to compare against the
values list
* @param {Array} valuesList - List of values to compare the attribute against
* @param {string} [partitionKeyName] - Optional name of the partition key attribute
for query operations
* @param {string} [partitionKeyValue] - Optional value of the partition key to
query
* @returns {Promise<Object>} - The response from DynamoDB containing the matching
items
*/
async function compareWithOrConditions(
  config,
  tableName,
  attributeName,
  valuesList,
  partitionKeyName,
  partitionKeyValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // If no values provided, return empty result
  if (!valuesList || valuesList.length === 0) {
    return {
      Items: [],
      Count: 0
    };
  }

  // Create the filter expression using multiple OR conditions
  const filterConditions = valuesList.map( (_, index) => `${attributeName} = :val${index}`);
  const filterExpression = filterConditions.join(' OR ');

  // Create expression attribute values for the values list
  const expressionAttributeValues = valuesList.reduce((acc, val, index) => {
    acc[` :val${index}`] = val;
    return acc;
  }, {});

  // If partition key is provided, perform a query operation
  if (partitionKeyName && partitionKeyValue) {
    const keyCondition = `${partitionKeyName} = :partitionKey`;

```

```
expressionAttributeValues[':partitionKey'] = partitionKeyValue;

// Initialize array to collect all items
let allItems = [];
let lastEvaluatedKey;

// Use pagination to get all results
do {
  const params = {
    TableName: tableName,
    KeyConditionExpression: keyCondition,
    FilterExpression: filterExpression,
    ExpressionAttributeValues: expressionAttributeValues
  };

  // Add ExclusiveStartKey if we have a lastEvaluatedKey from a previous query
  if (lastEvaluatedKey) {
    params.ExclusiveStartKey = lastEvaluatedKey;
  }

  const response = await docClient.send(new QueryCommand(params));

  // Add the items from this page to our collection
  if (response.Items && response.Items.length > 0) {
    allItems = [...allItems, ...response.Items];
  }

  // Get the key for the next page of results
  lastEvaluatedKey = response.LastEvaluatedKey;
} while (lastEvaluatedKey);

// Return the complete result
return {
  Items: allItems,
  Count: allItems.length
};
} else {
  // Otherwise, perform a scan operation
  // Initialize array to collect all items
  let allItems = [];
  let lastEvaluatedKey;

  // Use pagination to get all results
  do {
```

```
const params = {
  TableName: tableName,
  FilterExpression: filterExpression,
  ExpressionAttributeValues: expressionAttributeValues
};

// Add ExclusiveStartKey if we have a lastEvaluatedKey from a previous scan
if (lastEvaluatedKey) {
  params.ExclusiveStartKey = lastEvaluatedKey;
}

const response = await docClient.send(new ScanCommand(params));

// Add the items from this page to our collection
if (response.Items && response.Items.length > 0) {
  allItems = [...allItems, ...response.Items];
}

// Get the key for the next page of results
lastEvaluatedKey = response.LastEvaluatedKey;
} while (lastEvaluatedKey);

// Return the complete result
return {
  Items: allItems,
  Count: allItems.length
};
}
}
```

Beispiel für die Verwendung des Vergleichs mehrerer Werte mit AWS SDK für JavaScript.

```
/**
 * Example of how to use the compareMultipleValues function.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "Products";
  const attributeName = "Category";
  const valuesList = ["Electronics", "Computers", "Accessories"];
```

```
console.log(`Searching for products in any of these categories:
${valuesList.join(', ')}`);

try {
  // Using the IN operator (recommended approach)
  console.log("\nApproach 1: Using the IN operator");
  const response = await compareMultipleValues(
    config,
    tableName,
    attributeName,
    valuesList
  );

  console.log(`Found ${response.Count} products in the specified categories`);

  // Using multiple OR conditions (alternative approach)
  console.log("\nApproach 2: Using multiple OR conditions");
  const response2 = await compareWithOrConditions(
    config,
    tableName,
    attributeName,
    valuesList
  );

  console.log(`Found ${response2.Count} products in the specified categories`);

  // Example with a query operation
  console.log("\nQuerying a specific manufacturer's products in multiple
categories");
  const partitionKeyName = "Manufacturer";
  const partitionKeyValue = "Acme";

  const response3 = await compareMultipleValues(
    config,
    tableName,
    attributeName,
    valuesList,
    partitionKeyName,
    partitionKeyValue
  );

  console.log(`Found ${response3.Count} Acme products in the specified
categories`);
```

```
// Explain the benefits of using the IN operator
console.log("\nBenefits of using the IN operator:");
console.log("1. More concise expression compared to multiple OR conditions");
console.log("2. Better readability and maintainability");
console.log("3. Potentially better performance with large value lists");
console.log("4. Simpler code that's less prone to errors");
console.log("5. Easier to modify when adding or removing values");

} catch (error) {
  console.error("Error:", error);
}
}
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für JavaScript - API-Referenz.
 - [Query](#)
 - [Scan](#)

Bedingtes Aktualisieren der TTL eines Elements

Im folgenden Codebeispiel wird gezeigt, wie die TTL eines Elements bedingungsabhängig aktualisiert wird.

SDK für JavaScript (v3)

Aktualisieren Sie die TTL für ein vorhandenes DynamoDB-Element in einer Tabelle mit einer Bedingung.

```
import { DynamoDBClient, UpdateItemCommand } from "@aws-sdk/client-dynamodb";
import { marshall, unmarshall } from "@aws-sdk/util-dynamodb";

export const updateItemConditional = async (tableName, partitionKey, sortKey, region = 'us-east-1', newAttribute = 'default-value') => {
  const client = new DynamoDBClient({
    region: region,
    endpoint: `https://dynamodb.${region}.amazonaws.com`
  });

  const currentTime = Math.floor(Date.now() / 1000);

  const params = {
```

```

    TableName: tableName,
    Key: marshall({
      artist: partitionKey,
      album: sortKey
    }),
    UpdateExpression: "SET newAttribute = :newAttribute",
    ConditionExpression: "expireAt > :expiration",
    ExpressionAttributeValues: marshall({
      ':newAttribute': newAttribute,
      ':expiration': currentTime
    }),
    ReturnValues: "ALL_NEW"
  });

  try {
    const response = await client.send(new UpdateItemCommand(params));
    const responseData = unmarshall(response.Attributes);
    console.log("Item updated successfully: ", responseData);
    return responseData;
  } catch (error) {
    if (error.name === "ConditionalCheckFailedException") {
      console.log("Condition check failed: Item's 'expireAt' is expired.");
    } else {
      console.error("Error updating item: ", error);
    }
    throw error;
  }
};

// Example usage (commented out for testing)
// updateItemConditional('your-table-name', 'your-partition-key-value', 'your-sort-key-value');

```

- Einzelheiten zur API finden Sie [UpdateItem](#) in der AWS SDK für JavaScript API-Referenz.

Zählen von Ausdrucksoperatoren

Im folgenden Codebeispiel wird gezeigt, wie Ausdrucksoperatoren in DynamoDB gezählt werden.

- Machen Sie sich mit der Obergrenze von 300 Operatoren in DynamoDB vertraut.
- Zählen Sie Operatoren in komplexen Ausdrücken.

- Optimieren Sie Ausdrücke, um innerhalb der Grenzwerte zu bleiben.

SDK für JavaScript (v3)

Demonstrieren Sie das Zählen von Ausdrucksoperatoren mit AWS SDK für JavaScript.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  QueryCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Create a complex filter expression with a specified number of conditions.
 *
 * This function demonstrates how to generate a complex expression with
 * a specific number of operators to test the 300 operator limit.
 *
 * @param {number} conditionsCount - Number of conditions to include
 * @param {boolean} useAnd - Whether to use AND (true) or OR (false) between
 * conditions
 * @returns {Object} - Object containing the filter expression and attribute values
 */
function createComplexFilterExpression(conditionsCount, useAnd = true) {
  // Initialize the expression parts and attribute values
  const conditions = [];
  const expressionAttributeValues = {};

  // Generate the specified number of conditions
  for (let i = 0; i < conditionsCount; i++) {
    // Alternate between different comparison operators for variety
    let condition;
    const valueKey = `:val${i}`;

    switch (i % 5) {
      case 0:
        condition = `attribute${i} = ${valueKey}`;
        expressionAttributeValues[valueKey] = `value${i}`;
        break;
      case 1:
        condition = `attribute${i} > ${valueKey}`;
        expressionAttributeValues[valueKey] = i;
    }
  }
}
```

```

        break;
    case 2:
        condition = `attribute${i} < ${valueKey}`;
        expressionAttributeValues[valueKey] = i * 10;
        break;
    case 3:
        condition = `contains(attribute${i}, ${valueKey})`;
        expressionAttributeValues[valueKey] = `substring${i}`;
        break;
    case 4:
        condition = `attribute_exists(attribute${i})`;
        break;
    }

    conditions.push(condition);
}

// Join the conditions with AND or OR
const operator = useAnd ? " AND " : " OR ";
const filterExpression = conditions.join(operator);

// Calculate the operator count
// Each condition has 1 operator (=, >, <, contains, attribute_exists)
// Each AND or OR between conditions is 1 operator
const operatorCount = conditionsCount + (conditionsCount > 0 ? conditionsCount - 1 : 0);

return {
    filterExpression,
    expressionAttributeValues,
    operatorCount
};
}

/**
 * Create a complex update expression with a specified number of operations.
 *
 * This function demonstrates how to generate a complex update expression with
 * a specific number of operators to test the 300 operator limit.
 *
 * @param {number} operationsCount - Number of operations to include
 * @returns {Object} - Object containing the update expression and attribute values
 */
function createComplexUpdateExpression(operationsCount) {

```

```

// Initialize the expression parts and attribute values
const setOperations = [];
const expressionAttributeValues = {};

// Generate the specified number of SET operations
for (let i = 0; i < operationsCount; i++) {
  // Alternate between different types of SET operations
  let operation;
  const valueKey = `:val${i}`;

  switch (i % 3) {
    case 0:
      // Simple assignment (1 operator: =)
      operation = `attribute${i} = ${valueKey}`;
      expressionAttributeValues[valueKey] = `value${i}`;
      break;
    case 1:
      // Addition (2 operators: = and +)
      operation = `attribute${i} = attribute${i} + ${valueKey}`;
      expressionAttributeValues[valueKey] = i;
      break;
    case 2:
      // Conditional assignment with if_not_exists (2 operators: = and
if_not_exists)
      operation = `attribute${i} = if_not_exists(attribute${i}, ${valueKey})`;
      expressionAttributeValues[valueKey] = i * 10;
      break;
  }

  setOperations.push(operation);
}

// Create the update expression
const updateExpression = `SET ${setOperations.join(", ")}`;

// Calculate the operator count
// Each operation has 1-2 operators as noted above
let operatorCount = 0;
for (let i = 0; i < operationsCount; i++) {
  operatorCount += (i % 3 === 0) ? 1 : 2;
}

return {
  updateExpression,

```

```

        expressionAttributeValues,
        operatorCount
    };
}

/**
 * Test the operator limit by attempting an operation with a complex expression.
 *
 * This function demonstrates what happens when an expression approaches or
 * exceeds the 300 operator limit.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {number} operatorCount - Target number of operators to include
 * @returns {Promise<Object>} - Result of the operation attempt
 */
async function testOperatorLimit(
    config,
    tableName,
    key,
    operatorCount
) {
    // Initialize the DynamoDB client
    const client = new DynamoDBClient(config);
    const docClient = DynamoDBDocumentClient.from(client);

    // Create a complex update expression with the specified operator count
    const { updateExpression, expressionAttributeValues, operatorCount: actualCount } =
        createComplexUpdateExpression(Math.ceil(operatorCount / 1.5)); // Adjust to get
        close to target count

    console.log(`Generated update expression with approximately ${actualCount}
operators`);

    // Define the update parameters
    const params = {
        TableName: tableName,
        Key: key,
        UpdateExpression: updateExpression,
        ExpressionAttributeValues: expressionAttributeValues,
        ReturnValues: "UPDATED_NEW"
    };
};

```

```

try {
  // Attempt the update operation
  const response = await docClient.send(new UpdateCommand(params));
  return {
    success: true,
    message: `Operation succeeded with ${actualCount} operators`,
    data: response
  };
} catch (error) {
  // Check if the error is due to exceeding the operator limit
  if (error.name === "ValidationException" &&
    error.message.includes("too many operators")) {
    return {
      success: false,
      message: `Operation failed: ${error.message}`,
      operatorCount: actualCount
    };
  }

  // Return other errors
  return {
    success: false,
    message: `Operation failed: ${error.message}`,
    error
  };
}
}

/**
 * Break down a complex expression into multiple simpler operations.
 *
 * This function demonstrates how to handle expressions that would exceed
 * the 300 operator limit by breaking them into multiple operations.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {number} totalOperations - Total number of operations to perform
 * @returns {Promise<Object>} - Result of the operations
 */
async function breakDownComplexExpression(
  config,
  tableName,

```

```
key,
totalOperations
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Calculate how many operations we can safely include in each batch
  // Using 150 as a conservative limit (well below 300)
  const operationsPerBatch = 100;
  const batchCount = Math.ceil(totalOperations / operationsPerBatch);

  console.log(`Breaking down ${totalOperations} operations into ${batchCount}
batches`);

  const results = [];

  // Process each batch
  for (let batch = 0; batch < batchCount; batch++) {
    // Calculate the operations for this batch
    const batchStart = batch * operationsPerBatch;
    const batchEnd = Math.min(batchStart + operationsPerBatch, totalOperations);
    const batchSize = batchEnd - batchStart;

    console.log(`Processing batch ${batch + 1}/${batchCount} with ${batchSize}
operations`);

    // Create an update expression for this batch
    const { updateExpression, expressionAttributeValues, operatorCount } =
      createComplexUpdateExpression(batchSize);

    // Define the update parameters
    const params = {
      TableName: tableName,
      Key: key,
      UpdateExpression: updateExpression,
      ExpressionAttributeValues: expressionAttributeValues,
      ReturnValues: "UPDATED_NEW"
    };

    try {
      // Perform the update operation for this batch
      const response = await docClient.send(new UpdateCommand(params));
```

```
        results.push({
            batch: batch + 1,
            success: true,
            operatorCount,
            attributes: response.Attributes
        });
    } catch (error) {
        results.push({
            batch: batch + 1,
            success: false,
            operatorCount,
            error: error.message
        });

        // Stop processing if an error occurs
        break;
    }
}

return {
    totalBatches: batchCount,
    results
};
}

/**
 * Count operators in a DynamoDB expression based on the rules in the documentation.
 *
 * This function demonstrates how operators are counted according to the
 * DynamoDB documentation.
 *
 * @param {string} expression - The DynamoDB expression to analyze
 * @returns {Object} - Breakdown of operator counts
 */
function countOperatorsInExpression(expression) {
    // Initialize counters for different operator types
    const counts = {
        comparisonOperators: 0,
        logicalOperators: 0,
        functions: 0,
        arithmeticOperators: 0,
        specialOperators: 0,
        total: 0
    };
};
```

```
// Count comparison operators (=, <>, <, <=, >, >=)
const comparisonRegex = /^[^<>]=|^|=|^<|^>|^<=|^>=|^<[^>]>|^>[^<]<|^<[^>]<|^>[^<]>/g;
const comparisonMatches = expression.match(comparisonRegex) || [];
counts.comparisonOperators = comparisonMatches.length;

// Count logical operators (AND, OR, NOT)
const andMatches = expression.match(/\bAND\b/g) || [];
const orMatches = expression.match(/\bOR\b/g) || [];
const notMatches = expression.match(/\bNOT\b/g) || [];
counts.logicalOperators = andMatches.length + orMatches.length +
notMatches.length;

// Count functions (attribute_exists, attribute_not_exists, attribute_type,
begins_with, contains, size)
const functionRegex = /\b(attribute_exists|attribute_not_exists|attribute_type|
begins_with|contains|size|if_not_exists)\(/g;
const functionMatches = expression.match(functionRegex) || [];
counts.functions = functionMatches.length;

// Count arithmetic operators (+ and -)
const arithmeticMatches = expression.match(/[a-zA-Z0-9_]\s*[\+|-]\s*[a-zA-Z0-9_(:]/g) || [];
counts.arithmeticOperators = arithmeticMatches.length;

// Count special operators (BETWEEN, IN)
const betweenMatches = expression.match(/\bBETWEEN\b/g) || [];
const inMatches = expression.match(/\bIN\b/g) || [];
counts.specialOperators = betweenMatches.length + inMatches.length;

// Add extra operators for BETWEEN (each BETWEEN includes an AND)
counts.logicalOperators += betweenMatches.length;

// Calculate total
counts.total = counts.comparisonOperators +
counts.logicalOperators +
counts.functions +
counts.arithmeticOperators +
counts.specialOperators;

return counts;
}
```


Beispiel für die Verwendung des Ausdrucksoperators Counting with AWS SDK für JavaScript.

```
/**
 * Example of how to work with expression operator counting.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "Products";
  const key = { ProductId: "P12345" };

  console.log("Demonstrating DynamoDB expression operator counting and the 300
operator limit");

  try {
    // Example 1: Analyze a simple expression
    console.log("\nExample 1: Analyzing a simple expression");
    const simpleExpression = "Price = :price AND Rating > :rating AND Category IN
(:cat1, :cat2, :cat3)";
    const simpleCount = countOperatorsInExpression(simpleExpression);

    console.log(`Expression: ${simpleExpression}`);
    console.log("Operator count breakdown:");
    console.log(`- Comparison operators: ${simpleCount.comparisonOperators}`);
    console.log(`- Logical operators: ${simpleCount.logicalOperators}`);
    console.log(`- Functions: ${simpleCount.functions}`);
    console.log(`- Arithmetic operators: ${simpleCount.arithmeticOperators}`);
    console.log(`- Special operators: ${simpleCount.specialOperators}`);
    console.log(`- Total operators: ${simpleCount.total}`);

    // Example 2: Analyze a complex expression
    console.log("\nExample 2: Analyzing a complex expression");
    const complexExpression =
      "(attribute_exists(Category) AND Size BETWEEN :min AND :max) OR " +
      "(Price > :price AND contains(Description, :keyword) AND " +
      "(Rating >= :minRating OR Reviews > :minReviews))";
    const complexCount = countOperatorsInExpression(complexExpression);

    console.log(`Expression: ${complexExpression}`);
    console.log("Operator count breakdown:");
    console.log(`- Comparison operators: ${complexCount.comparisonOperators}`);
    console.log(`- Logical operators: ${complexCount.logicalOperators}`);
    console.log(`- Functions: ${complexCount.functions}`);
    console.log(`- Arithmetic operators: ${complexCount.arithmeticOperators}`);
```

```
console.log(`- Special operators: ${complexCount.specialOperators}`);
console.log(`- Total operators: ${complexCount.total}`);

// Example 3: Test approaching the operator limit
console.log("\nExample 3: Testing an expression approaching the operator
limit");
const approachingLimit = await testOperatorLimit(config, tableName, key, 290);
console.log(approachingLimit.message);

// Example 4: Test exceeding the operator limit
console.log("\nExample 4: Testing an expression exceeding the operator limit");
const exceedingLimit = await testOperatorLimit(config, tableName, key, 310);
console.log(exceedingLimit.message);

// Example 5: Breaking down a complex expression
console.log("\nExample 5: Breaking down a complex expression into multiple
operations");
const breakdownResult = await breakDownComplexExpression(config, tableName, key,
500);
console.log(`Processed ${breakdownResult.results.length} of
${breakdownResult.totalBatches} batches`);

// Explain the operator counting rules
console.log("\nKey points about DynamoDB expression operator counting:");
console.log("1. The maximum number of operators in any expression is 300");
console.log("2. Each comparison operator (=, <>, <, <=, >, >=) counts as 1
operator");
console.log("3. Each logical operator (AND, OR, NOT) counts as 1 operator");
console.log("4. Each function call (attribute_exists, contains, etc.) counts as
1 operator");
console.log("5. Each arithmetic operator (+ or -) counts as 1 operator");
console.log("6. BETWEEN counts as 2 operators (BETWEEN itself and the AND within
it)");
console.log("7. IN counts as 1 operator regardless of the number of values");
console.log("8. Parentheses for grouping and attribute paths don't count as
operators");
console.log("9. When you exceed the limit, the error always reports '301
operators'");
console.log("10. For complex operations, break them into multiple smaller
operations");

} catch (error) {
  console.error("Error:", error);
}
```

```
}
```

- Einzelheiten zur API finden Sie [UpdateItem](#) unter AWS SDK für JavaScript API-Referenz.

Erstellen einer Serverless-Anwendung zur Verwaltung von Fotos

Das folgende Codebeispiel zeigt, wie eine Serverless-Anwendung erstellt wird, mit der Benutzer Fotos mithilfe von Labels erstellen können.

SDK für JavaScript (v3)

Zeigt, wie eine Anwendung zur Verwaltung von Fotobeständen entwickelt wird, die mithilfe von Amazon Rekognition Labels in Bildern erkennt und sie für einen späteren Abruf speichert.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

Einen tiefen Einblick in den Ursprung dieses Beispiels finden Sie im Beitrag in der [AWS - Community](#).

In diesem Beispiel verwendete Dienste

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Erstellen einer Tabelle mit aktiviertem Warmdurchsatz

Im folgenden Codebeispiel wird gezeigt, wie eine Tabelle mit aktiviertem Warmdurchsatz erstellt wird.

SDK für JavaScript (v3)

Erstellen Sie eine DynamoDB-Tabelle mit Warmdurchsatz-Einstellung mithilfe von AWS SDK für JavaScript.

```
import { DynamoDBClient, CreateTableCommand } from "@aws-sdk/client-dynamodb";
```

```
export async function createDynamoDBTableWithWarmThroughput(
  tableName,
  partitionKey,
  sortKey,
  miscKeyAttr,
  nonKeyAttr,
  tableProvisionedReadUnits,
  tableProvisionedWriteUnits,
  tableWarmReads,
  tableWarmWrites,
  indexName,
  indexProvisionedReadUnits,
  indexProvisionedWriteUnits,
  indexWarmReads,
  indexWarmWrites,
  region = "us-east-1"
) {
  try {
    const ddbClient = new DynamoDBClient({ region: region });
    const command = new CreateTableCommand({
      TableName: tableName,
      AttributeDefinitions: [
        { AttributeName: partitionKey, AttributeType: "S" },
        { AttributeName: sortKey, AttributeType: "S" },
        { AttributeName: miscKeyAttr, AttributeType: "N" },
      ],
      KeySchema: [
        { AttributeName: partitionKey, KeyType: "HASH" },
        { AttributeName: sortKey, KeyType: "RANGE" },
      ],
      ProvisionedThroughput: {
        ReadCapacityUnits: tableProvisionedReadUnits,
        WriteCapacityUnits: tableProvisionedWriteUnits,
      },
      WarmThroughput: {
        ReadUnitsPerSecond: tableWarmReads,
        WriteUnitsPerSecond: tableWarmWrites,
      },
      GlobalSecondaryIndexes: [
        {
          IndexName: indexName,
          KeySchema: [
            { AttributeName: sortKey, KeyType: "HASH" },

```

```

        { AttributeName: miscKeyAttr, KeyType: "RANGE" },
      ],
      Projection: {
        ProjectionType: "INCLUDE",
        NonKeyAttributes: [nonKeyAttr],
      },
      ProvisionedThroughput: {
        ReadCapacityUnits: indexProvisionedReadUnits,
        WriteCapacityUnits: indexProvisionedWriteUnits,
      },
      WarmThroughput: {
        ReadUnitsPerSecond: indexWarmReads,
        WriteUnitsPerSecond: indexWarmWrites,
      },
    ],
  });
  const response = await ddbClient.send(command);
  console.log(response);
  return response;
} catch (error) {
  console.error(`Error creating table: ${error}`);
  throw error;
}
}

// Example usage (commented out for testing)
/*
createDynamoDBTableWithWarmThroughput(
  'example-table',
  'pk',
  'sk',
  'gsiKey',
  'data',
  10, 10, 5, 5,
  'example-index',
  5, 5, 2, 2
);
*/

```

- Einzelheiten zur API finden Sie [CreateTable](#) in der AWS SDK für JavaScript API-Referenz.

Erstellen eines Elements mit einer TTL

Das folgende Codebeispiel zeigt, wie Sie ein Element mit TTL erstellen.

SDK für JavaScript (v3)

```
import { DynamoDBClient, PutItemCommand } from "@aws-sdk/client-dynamodb";

export function createDynamoDBItem(table_name, region, partition_key, sort_key) {
  const client = new DynamoDBClient({
    region: region,
    endpoint: `https://dynamodb.${region}.amazonaws.com`
  });

  // Get the current time in epoch second format
  const current_time = Math.floor(new Date().getTime() / 1000);

  // Calculate the expireAt time (90 days from now) in epoch second format
  const expire_at = Math.floor((new Date().getTime() + 90 * 24 * 60 * 60 * 1000) /
1000);

  // Create DynamoDB item
  const item = {
    'partitionKey': {'S': partition_key},
    'sortKey': {'S': sort_key},
    'createdAt': {'N': current_time.toString()},
    'expireAt': {'N': expire_at.toString()}
  };

  const putItemCommand = new PutItemCommand({
    TableName: table_name,
    Item: item,
    ProvisionedThroughput: {
      ReadCapacityUnits: 1,
      WriteCapacityUnits: 1,
    },
  });

  client.send(putItemCommand, function(err, data) {
    if (err) {
      console.log("Exception encountered when creating item %s, here's what
happened: ", data, err);
      throw err;
    } else {
```

```

        console.log("Item created successfully: %s.", data);
        return data;
    }
    });
}

// Example usage (commented out for testing)
// createDynamoDBItem('your-table-name', 'us-east-1', 'your-partition-key-value',
// 'your-sort-key-value');

```

- Einzelheiten zur API finden Sie [PutItem](#) in der AWS SDK für JavaScript API-Referenz.

Löschen von Daten mit PartiQL DELETE

Im folgenden Codebeispiel wird gezeigt, wie Sie Daten mit PartiQL-DELETE-Anweisungen löschen.

SDK für JavaScript (v3)

Löschen Sie Elemente aus einer DynamoDB-Tabelle mithilfe von PartiQL DELETE-Anweisungen mit AWS SDK für JavaScript

```

/**
 * This example demonstrates how to delete items from a DynamoDB table using
 * PartiQL.
 * It shows different ways to delete documents with various index types.
 */
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
    DynamoDBDocumentClient,
    ExecuteStatementCommand,
    BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

/**
 * Delete a single item by its partition key using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @returns The response from the ExecuteStatementCommand
 */
export const deleteItemByPartitionKey = async (

```

```

    tableName: string,
    partitionKeyName: string,
    partitionKeyValue: string | number
) => {
    const client = new DynamoDBClient({});
    const docClient = DynamoDBDocumentClient.from(client);

    const params = {
        Statement: `DELETE FROM "${tableName}" WHERE ${partitionKeyName} = ?`,
        Parameters: [partitionKeyValue],
    };

    try {
        const data = await docClient.send(new ExecuteStatementCommand(params));
        console.log("Item deleted successfully");
        return data;
    } catch (err) {
        console.error("Error deleting item:", err);
        throw err;
    }
};

/**
 * Delete an item by its composite key (partition key + sort key) using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @param sortKeyName - The name of the sort key attribute
 * @param sortKeyValue - The value of the sort key
 * @returns The response from the ExecuteStatementCommand
 */
export const deleteItemByCompositeKey = async (
    tableName: string,
    partitionKeyName: string,
    partitionKeyValue: string | number,
    sortKeyName: string,
    sortKeyValue: string | number
) => {
    const client = new DynamoDBClient({});
    const docClient = DynamoDBDocumentClient.from(client);

    const params = {

```



```

    Statement: `DELETE FROM "${tableName}" WHERE ${partitionKeyName} = ? AND
    ${sortKeyName} = ?`,
    Parameters: [partitionKeyValue, sortKeyValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item deleted successfully");
    return data;
  } catch (err) {
    console.error("Error deleting item:", err);
    throw err;
  }
};

/**
 * Delete an item with a condition to ensure the delete only happens if a condition
 * is met.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @param conditionAttribute - The attribute to check in the condition
 * @param conditionValue - The value to compare against in the condition
 * @returns The response from the ExecuteStatementCommand
 */
export const deleteItemWithCondition = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number,
  conditionAttribute: string,
  conditionValue: any
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `DELETE FROM "${tableName}" WHERE ${partitionKeyName} = ? AND
    ${conditionAttribute} = ?`,
    Parameters: [partitionKeyValue, conditionValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));

```

```
    console.log("Item deleted with condition successfully");
    return data;
  } catch (err) {
    console.error("Error deleting item with condition:", err);
    throw err;
  }
};

/**
 * Batch delete multiple items using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param keys - Array of objects containing key information
 * @returns The response from the BatchExecuteStatementCommand
 */
export const batchDeleteItems = async (
  tableName: string,
  keys: Array<{
    partitionKeyName: string;
    partitionKeyValue: string | number;
    sortKeyName?: string;
    sortKeyValue?: string | number;
  }>
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  // Create statements for each delete
  const statements = keys.map((key) => {
    if (key.sortKeyName && key.sortKeyValue !== undefined) {
      return {
        Statement: `DELETE FROM "${tableName}" WHERE ${key.partitionKeyName} = ? AND
${key.sortKeyName} = ?`,
        Parameters: [key.partitionKeyValue, key.sortKeyValue],
      };
    } else {
      return {
        Statement: `DELETE FROM "${tableName}" WHERE ${key.partitionKeyName} = ?`,
        Parameters: [key.partitionKeyValue],
      };
    }
  });

  const params = {
```

```
    Statements: statements,
  };

  try {
    const data = await docClient.send(new BatchExecuteStatementCommand(params));
    console.log("Items batch deleted successfully");
    return data;
  } catch (err) {
    console.error("Error batch deleting items:", err);
    throw err;
  }
};

/**
 * Delete multiple items that match a filter condition.
 * Note: This performs a scan operation which can be expensive on large tables.
 *
 * @param tableName - The name of the DynamoDB table
 * @param filterAttribute - The attribute to filter on
 * @param filterValue - The value to filter by
 * @returns The response from the ExecuteStatementCommand
 */
export const deleteItemsByFilter = async (
  tableName: string,
  filterAttribute: string,
  filterValue: any
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `DELETE FROM "${tableName}" WHERE ${filterAttribute} = ?`,
    Parameters: [filterValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Items deleted by filter successfully");
    return data;
  } catch (err) {
    console.error("Error deleting items by filter:", err);
    throw err;
  }
};
```

```
/**
 * Example usage showing how to delete items with different index types
 */
export const deleteExamples = async () => {
  // Delete an item by partition key (simple primary key)
  await deleteItemByPartitionKey("UsersTable", "userId", "user123");

  // Delete an item by composite key (partition key + sort key)
  await deleteItemByCompositeKey(
    "OrdersTable",
    "orderId",
    "order456",
    "productId",
    "prod789"
  );

  // Delete with a condition
  await deleteItemWithCondition(
    "UsersTable",
    "userId",
    "user789",
    "userStatus",
    "inactive"
  );

  // Batch delete multiple items
  await batchDeleteItems("UsersTable", [
    { partitionKeyName: "userId", partitionKeyValue: "user234" },
    { partitionKeyName: "userId", partitionKeyValue: "user345" },
  ]);

  // Batch delete items with composite keys
  await batchDeleteItems("OrdersTable", [
    {
      partitionKeyName: "orderId",
      partitionKeyValue: "order567",
      sortKeyName: "productId",
      sortKeyValue: "prod123",
    },
    {
      partitionKeyName: "orderId",
      partitionKeyValue: "order678",
      sortKeyName: "productId",
    }
  ]);
}
```

```

        sortKeyValue: "prod456",
    },
  ]);

  // Delete items by filter (use with caution)
  await deleteItemsByFilter("UsersTable", "userStatus", "deleted");
};

```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für JavaScript - API-Referenz.
 - [BatchExecuteStatement](#)
 - [ExecuteStatement](#)

Einfügen von Daten mithilfe von PartiQL INSERT

Im folgenden Codebeispiel wird gezeigt, wie Sie Daten mit PartiQL-INSERT-Anweisungen einfügen.

SDK für JavaScript (v3)

Fügen Sie Elemente mithilfe von PartiQL INSERT-Anweisungen mit in eine DynamoDB-Tabelle ein. AWS SDK für JavaScript

```

/**
 * This example demonstrates how to insert items into a DynamoDB table using
 * PartiQL.
 * It shows different ways to insert documents with various index types.
 */
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  ExecuteStatementCommand,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

/**
 * Insert a single item into a DynamoDB table using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param item - The item to insert
 * @returns The response from the ExecuteStatementCommand
 */

```

```
*/
export const insertItem = async (tableName: string, item: Record<string, any>) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  // Convert the item to a string representation for PartiQL
  const itemString = JSON.stringify(item).replace(/"([\^"]+)"/g, '$1:');

  const params = {
    Statement: `INSERT INTO "${tableName}" VALUE ${itemString}`,
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item inserted successfully");
    return data;
  } catch (err) {
    console.error("Error inserting item:", err);
    throw err;
  }
};

/**
 * Insert multiple items into a DynamoDB table using PartiQL batch operation.
 * This is more efficient than inserting items one by one.
 *
 * @param tableName - The name of the DynamoDB table
 * @param items - Array of items to insert
 * @returns The response from the BatchExecuteStatementCommand
 */
export const batchInsertItems = async (tableName: string, items: Record<string, any>[]) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  // Create statements for each item
  const statements = items.map((item) => {
    const itemString = JSON.stringify(item).replace(/"([\^"]+)"/g, '$1:');
    return {
      Statement: `INSERT INTO "${tableName}" VALUE ${itemString}`,
    };
  });

  const params = {
```

```

    Statements: statements,
  };

  try {
    const data = await docClient.send(new BatchExecuteStatementCommand(params));
    console.log("Items inserted successfully");
    return data;
  } catch (err) {
    console.error("Error batch inserting items:", err);
    throw err;
  }
};

/**
 * Insert an item with a condition to prevent overwriting existing items.
 * This is useful for ensuring you don't accidentally overwrite data.
 *
 * @param tableName - The name of the DynamoDB table
 * @param item - The item to insert
 * @param partitionKeyName - The name of the partition key attribute
 * @returns The response from the ExecuteStatementCommand
 */
export const insertItemWithCondition = async (
  tableName: string,
  item: Record<string, any>,
  partitionKeyName: string
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const itemString = JSON.stringify(item).replace(/"([^\"]+)":/g, '$1:');
  const partitionKeyValue = JSON.stringify(item[partitionKeyName]);

  const params = {
    Statement: `INSERT INTO "${tableName}" VALUE ${itemString} WHERE
attribute_not_exists(${partitionKeyName})`,
    Parameters: [{ S: partitionKeyValue }],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item inserted with condition successfully");
    return data;
  } catch (err) {

```

```
    console.error("Error inserting item with condition:", err);
    throw err;
  }
};

/**
 * Example usage showing how to insert items with different index types
 */
export const insertExamples = async () => {
  // Example table with a simple primary key (just partition key)
  const simpleKeyItem = {
    userId: "user123",
    name: "John Doe",
    email: "john@example.com",
  };
  await insertItem("UsersTable", simpleKeyItem);

  // Example table with composite key (partition key + sort key)
  const compositeKeyItem = {
    orderId: "order456",
    productId: "prod789",
    quantity: 2,
    price: 29.99,
  };
  await insertItem("OrdersTable", compositeKeyItem);

  // Example with Global Secondary Index (GSI)
  // The GSI might be on the email attribute
  const gsiItem = {
    userId: "user789",
    email: "jane@example.com",
    name: "Jane Smith",
    userType: "premium", // This could be part of a GSI
  };
  await insertItem("UsersTable", gsiItem);

  // Example with Local Secondary Index (LSI)
  // LSI uses the same partition key but different sort key
  const lsiItem = {
    orderId: "order567", // Partition key
    productId: "prod123", // Sort key for the table
    orderDate: "2023-11-15", // Potential sort key for an LSI
    quantity: 1,
    price: 19.99,
  };
  await insertItem("OrdersTable", lsiItem);
};
```



```
};  
await insertItem("OrdersTable", lsiItem);  
  
// Batch insert example with multiple items  
const batchItems = [  
  {  
    userId: "user234",  
    name: "Alice Johnson",  
    email: "alice@example.com",  
  },  
  {  
    userId: "user345",  
    name: "Bob Williams",  
    email: "bob@example.com",  
  },  
];  
await batchInsertItems("UsersTable", batchItems);  
};
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für JavaScript - API-Referenz.
 - [BatchExecuteStatement](#)
 - [ExecuteStatement](#)

Aufrufen einer Lambda-Funktion von einem Browser aus

Das folgende Codebeispiel zeigt, wie eine AWS Lambda Funktion von einem Browser aus aufgerufen wird.

SDK für JavaScript (v3)

Sie können eine browserbasierte Anwendung erstellen, die eine AWS Lambda Funktion verwendet, um eine Amazon DynamoDB-Tabelle mit Benutzerauswahlen zu aktualisieren. Diese App verwendet v3. AWS SDK für JavaScript

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

In diesem Beispiel verwendete Dienste

- DynamoDB

- Lambda

Ausführen erweiterter Abfrageoperationen

Im folgenden Codebeispiel wird gezeigt, wie Sie erweiterte Abfrageoperationen in DynamoDB ausführen.

- Abfragen von Tabellen mithilfe verschiedener Filter- und Bedingungstechniken
- Implementieren der Paginierung für große Ergebnismengen
- Verwenden globaler sekundärer Indizes für alternative Zugriffsmuster
- Anwenden von Konsistenzkontrollen auf Grundlage der Anwendungsanforderungen

SDK für JavaScript (v3)

Abfrage mit stark konsistenten Lesevorgängen unter Verwendung von AWS SDK für JavaScript.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table with configurable read consistency
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {boolean} useConsistentRead - Whether to use strongly consistent reads
 * @returns {Promise<Object>} - The query response
 */
async function queryWithConsistentRead(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  useConsistentRead = false
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input
    const input = {
```

```

    TableName: tableName,
    KeyConditionExpression: "#pk = :pkValue",
    ExpressionAttributeNames: {
        "#pk": partitionKeyName
    },
    ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue }
    },
    ConsistentRead: useConsistentRead
};

// Execute the query
const command = new QueryCommand(input);
return await client.send(command);
} catch (error) {
    console.error(`Error querying with consistent read: ${error}`);
    throw error;
}
}

```

Abfrage unter Verwendung eines globalen sekundären Indexes mit AWS SDK für JavaScript.

```

const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table using the primary key
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} userId - The user ID to query by (partition key)
 * @returns {Promise<Object>} - The query response
 */
async function queryTable(
    config,
    tableName,
    userId
) {
    try {
        // Create DynamoDB client
        const client = new DynamoDBClient(config);

        // Construct the query input for the base table
    }
}

```

```
const input = {
  TableName: tableName,
  KeyConditionExpression: "user_id = :userId",
  ExpressionAttributeValues: {
    ":userId": { S: userId }
  }
};

// Execute the query
const command = new QueryCommand(input);
return await client.send(command);
} catch (error) {
  console.error(`Error querying table: ${error}`);
  throw error;
}
}

/**
 * Queries a DynamoDB Global Secondary Index (GSI)
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} indexName - The name of the GSI to query
 * @param {string} gameId - The game ID to query by (GSI partition key)
 * @returns {Promise<Object>} - The query response
 */
async function queryGSI(
  config,
  tableName,
  indexName,
  gameId
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input for the GSI
    const input = {
      TableName: tableName,
      IndexName: indexName,
      KeyConditionExpression: "game_id = :gameId",
      ExpressionAttributeValues: {
        ":gameId": { S: gameId }
      }
    }
  }
}
```

```

    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying GSI: ${error}`);
    throw error;
  }
}

```

Abfrage mit Paginierung unter Verwendung von AWS SDK für JavaScript.

```

/**
 * Example demonstrating how to handle large query result sets in DynamoDB using
 * pagination
 *
 * This example shows:
 * - How to use pagination to handle large result sets
 * - How to use LastEvaluatedKey to retrieve the next page of results
 * - How to construct subsequent query requests using ExclusiveStartKey
 */
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table with pagination to handle large result sets
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {number} pageSize - Number of items per page
 * @returns {Promise<Array>} - All items from the query
 */
async function queryWithPagination(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  pageSize = 25
) {

```

```
try {
  // Create DynamoDB client
  const client = new DynamoDBClient(config);

  // Initialize variables for pagination
  let lastEvaluatedKey = undefined;
  const allItems = [];
  let pageCount = 0;

  // Loop until all pages are retrieved
  do {
    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue",
      Limit: pageSize,
      ExpressionAttributeNames: {
        "#pk": partitionKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue }
      }
    };

    // Add ExclusiveStartKey if we have a LastEvaluatedKey from a previous query
    if (lastEvaluatedKey) {
      input.ExclusiveStartKey = lastEvaluatedKey;
    }

    // Execute the query
    const command = new QueryCommand(input);
    const response = await client.send(command);

    // Process the current page of results
    pageCount++;
    console.log(`Processing page ${pageCount} with ${response.Items.length}
items`);

    // Add the items from this page to our collection
    if (response.Items && response.Items.length > 0) {
      allItems.push(...response.Items);
    }

    // Get the LastEvaluatedKey for the next page
```

```

        lastEvaluatedKey = response.LastEvaluatedKey;

    } while (lastEvaluatedKey); // Continue until there are no more pages

    console.log(`Query complete. Retrieved ${allItems.length} items in ${pageCount}
pages.`);
    return allItems;
} catch (error) {
    console.error(`Error querying with pagination: ${error}`);
    throw error;
}
}

/**
 * Example usage:
 *
 * // Query all items in the "AWS DynamoDB" forum with pagination
 * const allItems = await queryWithPagination(
 *   { region: "us-west-2" },
 *   "ForumThreads",
 *   "ForumName",
 *   "AWS DynamoDB",
 *   25 // 25 items per page
 * );
 *
 * console.log(`Total items retrieved: ${allItems.length}`);
 *
 * // Notes on pagination:
 * // - LastEvaluatedKey contains the primary key of the last evaluated item
 * // - When LastEvaluatedKey is undefined/null, there are no more items to retrieve
 * // - ExclusiveStartKey tells DynamoDB where to start the next page
 * // - Pagination helps manage memory usage for large result sets
 * // - Each page requires a separate network request to DynamoDB
 */

module.exports = { queryWithPagination };

```

Abfrage mit komplexen Filtern unter Verwendung von AWS SDK für JavaScript.

```

const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**

```

```
* Queries a DynamoDB table with a complex filter expression
*
* @param {Object} config - AWS SDK configuration object
* @param {string} tableName - The name of the DynamoDB table
* @param {string} partitionKeyName - The name of the partition key
* @param {string} partitionKeyValue - The value of the partition key
* @param {number|string} minViews - Minimum number of views for filtering
* @param {number|string} minReplies - Minimum number of replies for filtering
* @param {string} requiredTag - Tag that must be present in the item's tags set
* @returns {Promise<Object>} - The query response
*/
async function queryWithComplexFilter(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  minViews,
  minReplies,
  requiredTag
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue",
      FilterExpression: "views >= :minViews AND replies >= :minReplies AND
contains(tags, :tag)",
      ExpressionAttributeNames: {
        "#pk": partitionKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue },
        ":minViews": { N: minViews.toString() },
        ":minReplies": { N: minReplies.toString() },
        ":tag": { S: requiredTag }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  }
}
```



```
    } catch (error) {  
      console.error(`Error querying with complex filter: ${error}`);  
      throw error;  
    }  
  }  
}
```

Abfrage mit einem dynamisch erstellten Filterausdruck unter Verwendung von AWS SDK für JavaScript.

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");  
  
async function queryWithDynamicFilter(  
  config,  
  tableName,  
  partitionKeyName,  
  partitionKeyValue,  
  sortKeyName,  
  sortKeyValue,  
  filterParams = {}  
) {  
  try {  
    // Create DynamoDB client  
    const client = new DynamoDBClient(config);  
  
    // Initialize filter expression components  
    let filterExpressions = [];  
    const expressionAttributeValues = {  
      ":pkValue": { S: partitionKeyValue },  
      ":skValue": { S: sortKeyValue }  
    };  
    const expressionAttributeNames = {  
      "#pk": partitionKeyName,  
      "#sk": sortKeyName  
    };  
  
    // Add status filter if provided  
    if (filterParams.status) {  
      filterExpressions.push("status = :status");  
      expressionAttributeValues[":status"] = { S: filterParams.status };  
    }  
  
    // Add minimum views filter if provided
```

```
    if (filterParams.minViews !== undefined) {
      filterExpressions.push("views >= :minViews");
      expressionAttributeValues[":minViews"] = { N:
filterParams.minViews.toString() };
    }

    // Add author filter if provided
    if (filterParams.author) {
      filterExpressions.push("author = :author");
      expressionAttributeValues[":author"] = { S: filterParams.author };
    }

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue AND #sk = :skValue"
    };

    // Add filter expression if any filters were provided
    if (filterExpressions.length > 0) {
      input.FilterExpression = filterExpressions.join(" AND ");
    }

    // Add expression attribute names and values
    input.ExpressionAttributeNames = expressionAttributeNames;
    input.ExpressionAttributeValues = expressionAttributeValues;

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with dynamic filter: ${error}`);
    throw error;
  }
}
```

- Weitere API-Informationen finden Sie unter [Abfragen](#) in der AWS SDK für JavaScript -API-Referenz.

Ausführen von Listenoperationen

Das folgende Codebeispiel zeigt, wie Listenoperationen für DynamoDB-Tabellen ausgeführt werden.

- Hinzufügen von Elementen zu einem Listenattribut
- Entfernen von Elementen aus einem Listenattribut
- Aktualisieren von bestimmten Elementen in einer Liste nach Index
- Verwenden von Funktionen zum Anfügen und zum Auflisten von Indizes

SDK für JavaScript (v3)

Demonstrieren Sie Listenoperationen mit AWS SDK für JavaScript.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  GetCommand,
  PutCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Append elements to a list attribute.
 *
 * This function demonstrates how to use the list_append function to add elements
 * to the end of a list.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName - The name of the list attribute
 * @param {Array} values - The values to append to the list
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function appendToList(
  config,
  tableName,
  key,
  listName,
  values
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using list_append
```

```

const params = {
  TableName: tableName,
  Key: key,
  UpdateExpression: `SET ${listName} =
list_append(if_not_exists(${listName}, :empty_list), :values)`,
  ExpressionAttributeValues: {
    ":empty_list": [],
    ":values": values
  },
  ReturnValues: "UPDATED_NEW"
};

// Perform the update operation
const response = await docClient.send(new UpdateCommand(params));

return response;
}

/**
 * Prepend elements to a list attribute.
 *
 * This function demonstrates how to use the list_append function to add elements
 * to the beginning of a list.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName - The name of the list attribute
 * @param {Array} values - The values to prepend to the list
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function prependToList(
  config,
  tableName,
  key,
  listName,
  values
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using list_append
  // Note: To prepend, we put the new values first in the list_append function

```

```

    const params = {
      TableName: tableName,
      Key: key,
      UpdateExpression: `SET ${listName} = list_append(:values,
if_not_exists(${listName}, :empty_list))`,
      ExpressionAttributeValues: {
        ":empty_list": [],
        ":values": values
      },
      ReturnValues: "UPDATED_NEW"
    };

    // Perform the update operation
    const response = await docClient.send(new UpdateCommand(params));

    return response;
  }

/**
 * Update a specific element in a list by index.
 *
 * This function demonstrates how to update a specific element in a list
 * using the index notation.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName - The name of the list attribute
 * @param {number} index - The index of the element to update
 * @param {any} value - The new value for the element
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateListElement(
  config,
  tableName,
  key,
  listName,
  index,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

```

```

// Define the update parameters using index notation
const params = {
  TableName: tableName,
  Key: key,
  UpdateExpression: `SET ${listName}[${index}] = :value`,
  ExpressionAttributeValues: {
    ":value": value
  },
  ReturnValues: "UPDATED_NEW"
};

// Perform the update operation
const response = await docClient.send(new UpdateCommand(params));

return response;
}

/**
 * Remove an element from a list by index.
 *
 * This function demonstrates how to remove a specific element from a list
 * using the REMOVE action with index notation.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName - The name of the list attribute
 * @param {number} index - The index of the element to remove
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function removeListElement(
  config,
  tableName,
  key,
  listName,
  index
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using REMOVE with index notation
  const params = {
    TableName: tableName,

```

```

    Key: key,
    UpdateExpression: `REMOVE ${listName}[${index}]`,
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Concatenate two lists.
 *
 * This function demonstrates how to concatenate two lists using the list_append
 * function.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName1 - The name of the first list attribute
 * @param {string} listName2 - The name of the second list attribute
 * @param {string} resultListName - The name of the attribute to store the
 * concatenated list
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function concatenateLists(
  config,
  tableName,
  key,
  listName1,
  listName2,
  resultListName
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using list_append
  const params = {
    TableName: tableName,
    Key: key,

```

```

    UpdateExpression: `SET ${resultListName} =
list_append(if_not_exists(${listName1}, :empty_list),
if_not_exists(${listName2}, :empty_list))`,
    ExpressionAttributeValues: {
        ":empty_list": []
    },
    ReturnValues: "UPDATED_NEW"
};

// Perform the update operation
const response = await docClient.send(new UpdateCommand(params));

return response;
}

/**
 * Create a nested list structure.
 *
 * This function demonstrates how to create and work with nested lists.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName - The name of the list attribute
 * @param {Array} nestedLists - An array of arrays to create a nested list structure
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function createNestedList(
    config,
    tableName,
    key,
    listName,
    nestedLists
) {
    // Initialize the DynamoDB client
    const client = new DynamoDBClient(config);
    const docClient = DynamoDBDocumentClient.from(client);

    // Define the update parameters to create a nested list
    const params = {
        TableName: tableName,
        Key: key,
        UpdateExpression: `SET ${listName} = :nested_lists`,
        ExpressionAttributeValues: {

```



```

        ":nested_lists": nestedLists
    },
    ReturnValues: "UPDATED_NEW"
};

// Perform the update operation
const response = await docClient.send(new UpdateCommand(params));

return response;
}

/**
 * Update an element in a nested list.
 *
 * This function demonstrates how to update an element in a nested list
 * using multiple index notations.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} listName - The name of the list attribute
 * @param {number} outerIndex - The index in the outer list
 * @param {number} innerIndex - The index in the inner list
 * @param {any} value - The new value for the element
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateNestedListElement(
    config,
    tableName,
    key,
    listName,
    outerIndex,
    innerIndex,
    value
) {
    // Initialize the DynamoDB client
    const client = new DynamoDBClient(config);
    const docClient = DynamoDBDocumentClient.from(client);

    // Define the update parameters using multiple index notations
    const params = {
        TableName: tableName,
        Key: key,
        UpdateExpression: `SET ${listName}[${outerIndex}][${innerIndex}] = :value`,
    };

```

```
    ExpressionAttributeValues: {
      ":value": value
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Get the current value of an item.
 *
 * Helper function to retrieve the current value of an item.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to get
 * @returns {Promise<Object|null>} - The item or null if not found
 */
async function getItem(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the get parameters
  const params = {
    TableName: tableName,
    Key: key
  };

  // Perform the get operation
  const response = await docClient.send(new GetCommand(params));

  // Return the item if it exists, otherwise null
  return response.Item || null;
}
```

Beispiel für die Verwendung von Listenoperationen mit AWS SDK für JavaScript.

```
/**
 * Example of how to work with lists in DynamoDB.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "UserProfiles";
  const key = { UserId: "U12345" };

  console.log("Demonstrating list operations in DynamoDB");

  try {
    // Example 1: Append elements to a list
    console.log("\nExample 1: Appending elements to a list");
    const response1 = await appendToList(
      config,
      tableName,
      key,
      "RecentSearches",
      ["laptop", "headphones", "monitor"]
    );

    console.log("Appended to list:", response1.Attributes);

    // Example 2: Prepend elements to a list
    console.log("\nExample 2: Prepending elements to a list");
    const response2 = await prependToList(
      config,
      tableName,
      key,
      "RecentSearches",
      ["keyboard", "mouse"]
    );

    console.log("Prepended to list:", response2.Attributes);

    // Get the current state of the item
    let currentItem = await getItem(config, tableName, key);
    console.log("\nCurrent state of RecentSearches:", currentItem?.RecentSearches);
  }
}
```

```
// Example 3: Update a specific element in a list
console.log("\nExample 3: Updating a specific element in a list");
const response3 = await updateListElement(
  config,
  tableName,
  key,
  "RecentSearches",
  0, // Update the first element
  "mechanical keyboard" // New value
);

console.log("Updated list element:", response3.Attributes);

// Example 4: Remove an element from a list
console.log("\nExample 4: Removing an element from a list");
const response4 = await removeListElement(
  config,
  tableName,
  key,
  "RecentSearches",
  2 // Remove the third element
);

console.log("List after removing element:", response4.Attributes);

// Example 5: Create and concatenate lists
console.log("\nExample 5: Creating and concatenating lists");

// First, create two separate lists
await updateWithMultipleActions(
  config,
  tableName,
  key,
  "SET WishList = :wishlist, SavedItems = :saveditems",
  null,
  {
    ":wishlist": ["gaming laptop", "wireless earbuds"],
    ":saveditems": ["smartphone", "tablet"]
  }
);

// Then, concatenate them
const response5 = await concatenateLists(
```

```
    config,
    tableName,
    key,
    "WishList",
    "SavedItems",
    "AllItems"
  );

  console.log("Concatenated lists:", response5.Attributes);

  // Example 6: Create a nested list structure
  console.log("\nExample 6: Creating a nested list structure");
  const response6 = await createNestedList(
    config,
    tableName,
    key,
    "Categories",
    [
      ["Electronics", "Computers", "Accessories"],
      ["Books", "Magazines", "E-books"],
      ["Clothing", "Shoes", "Watches"]
    ]
  );

  console.log("Created nested list:", response6.Attributes);

  // Example 7: Update an element in a nested list
  console.log("\nExample 7: Updating an element in a nested list");
  const response7 = await updateNestedListElement(
    config,
    tableName,
    key,
    "Categories",
    0, // First inner list
    1, // Second element in that list
    "Laptops" // New value
  );

  console.log("Updated nested list element:", response7.Attributes);

  // Get the final state of the item
  currentItem = await getItem(config, tableName, key);
  console.log("\nFinal state of the item:", JSON.stringify(currentItem, null, 2));
```

```

    // Explain list operations
    console.log("\nKey points about list operations in DynamoDB:");
    console.log("1. Use list_append to add elements to a list");
    console.log("2. To append elements, use list_append(existingList,
newElements)");
    console.log("3. To prepend elements, use list_append(newElements,
existingList)");
    console.log("4. Use if_not_exists to handle cases where the list might not exist
yet");
    console.log("5. Use index notation (list[0]) to access or update specific
elements");
    console.log("6. Use REMOVE with index notation to remove elements from a list");
    console.log("7. Lists can contain elements of different types");
    console.log("8. Lists can be nested (lists of lists)");
    console.log("9. Use multiple index notations (list[0][1]) to access nested list
elements");

    } catch (error) {
        console.error("Error:", error);
    }
}

/**
 * Helper function for the examples.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} updateExpression - The update expression
 * @param {Object} expressionAttributeNames - Expression attribute name placeholders
 * @param {Object} expressionAttributeValues - Expression attribute value
placeholders
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateWithMultipleActions(
    config,
    tableName,
    key,
    updateExpression,
    expressionAttributeNames,
    expressionAttributeValues
) {
    // Initialize the DynamoDB client
    const client = new DynamoDBClient(config);

```

```
const docClient = DynamoDBDocumentClient.from(client);

// Prepare the update parameters
const updateParams = {
  TableName: tableName,
  Key: key,
  UpdateExpression: updateExpression,
  ReturnValues: "UPDATED_NEW"
};

// Add expression attribute names if provided
if (expressionAttributeNames) {
  updateParams.ExpressionAttributeNames = expressionAttributeNames;
}

// Add expression attribute values if provided
if (expressionAttributeValues) {
  updateParams.ExpressionAttributeValues = expressionAttributeValues;
}

// Execute the update
const response = await docClient.send(new UpdateCommand(updateParams));

return response;
}
```

- Einzelheiten zur API finden Sie [UpdateItem](#) in der AWS SDK für JavaScript API-Referenz.

Ausführen von Map-Operationen

Im folgenden Codebeispiel wird gezeigt, wie Map-Operationen in DynamoDB ausgeführt werden.

- Fügen Sie verschachtelte Attribute in Map-Strukturen hinzu und aktualisieren Sie sie.
- Entfernen bestimmter Felder aus Maps
- Arbeiten mit tief verschachtelten Map-Attributen

SDK für JavaScript (v3)

Demonstrieren Sie Kartenoperationen mit AWS SDK für JavaScript.

```
/**
 * Example of updating map attributes in DynamoDB.
 *
 * This module demonstrates how to update map attributes that may not exist,
 * how to update nested attributes, and how to handle various map update scenarios.
 */

const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  GetCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Update a map attribute safely, handling the case where the map might not exist.
 *
 * This function demonstrates using the if_not_exists function to safely update
 * a map attribute that might not exist yet.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} mapName - The name of the map attribute
 * @param {string} mapKey - The key within the map to update
 * @param {any} value - The value to set
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateMapAttributeSafe(
  config,
  tableName,
  key,
  mapName,
  mapKey,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using SET with if_not_exists
  const params = {
```



```

    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${mapName}.${mapKey} = :value`,
    ExpressionAttributeValues: {
      ":value": value
    },
    ReturnValues: "UPDATED_NEW"
  };

  try {
    // Perform the update operation
    const response = await docClient.send(new UpdateCommand(params));
    return response;
  } catch (error) {
    // If the error is because the map doesn't exist, create it
    if (error.name === "ValidationException" &&
        error.message.includes("The document path provided in the update expression
is invalid")) {

      // Create the map with the specified key-value pair
      const createParams = {
        TableName: tableName,
        Key: key,
        UpdateExpression: `SET ${mapName} = :map`,
        ExpressionAttributeValues: {
          ":map": { [mapKey]: value }
        },
        ReturnValues: "UPDATED_NEW"
      };

      return await docClient.send(new UpdateCommand(createParams));
    }

    // Re-throw other errors
    throw error;
  }
}

/**
 * Update a map attribute using the if_not_exists function.
 *
 * This function demonstrates a more elegant approach using if_not_exists
 * to handle the case where the map doesn't exist yet.
 */

```

```

* @param {Object} config - AWS configuration object
* @param {string} tableName - The name of the DynamoDB table
* @param {Object} key - The key of the item to update
* @param {string} mapName - The name of the map attribute
* @param {string} mapKey - The key within the map to update
* @param {any} value - The value to set
* @returns {Promise<Object>} - The response from DynamoDB
*/
async function updateMapAttributeWithIfNotExists(
  config,
  tableName,
  key,
  mapName,
  mapKey,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using SET with if_not_exists
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${mapName} = if_not_exists(${mapName}, :emptyMap),
${mapName}.${mapKey} = :value`,
    ExpressionAttributeValues: {
      ":emptyMap": {},
      ":value": value
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Add a value to a deeply nested map, creating parent maps if they don't exist.
 *
 * This function demonstrates how to update a deeply nested attribute,
 * creating any parent maps that don't exist along the way.

```

```

*
* @param {Object} config - AWS configuration object
* @param {string} tableName - The name of the DynamoDB table
* @param {Object} key - The key of the item to update
* @param {string[]} path - The path to the nested attribute as an array of keys
* @param {any} value - The value to set
* @returns {Promise<Object>} - The response from DynamoDB
*/
async function addToNestedMap(
  config,
  tableName,
  key,
  path,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Build the update expression and expression attribute values
  let updateExpression = "SET";
  const expressionAttributeValues = {};

  // For each level in the path, create a map if it doesn't exist
  for (let i = 0; i < path.length; i++) {
    const currentPath = path.slice(0, i + 1).join(".");
    const parentPath = i > 0 ? path.slice(0, i).join(".") : null;

    if (parentPath) {
      updateExpression += ` ${parentPath} = if_not_exists(${parentPath}, :emptyMap${i}),`;
      expressionAttributeValues[`:emptyMap${i}`] = {};
    }
  }

  // Set the final value
  const fullPath = path.join(".");
  updateExpression += ` ${fullPath} = :value`;
  expressionAttributeValues[`:value`] = value;

  // Define the update parameters
  const params = {
    TableName: tableName,
    Key: key,

```

```

    UpdateExpression: updateExpression,
    ExpressionAttributeValues: expressionAttributeValues,
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Update multiple fields in a map attribute in a single operation.
 *
 * This function demonstrates how to update multiple fields in a map
 * in a single DynamoDB operation.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} mapName - The name of the map attribute
 * @param {Object} updates - Object containing key-value pairs to update
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateMultipleMapFields(
  config,
  tableName,
  key,
  mapName,
  updates
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Build the update expression and expression attribute values
  let updateExpression = `SET ${mapName} = if_not_exists(${mapName}, :emptyMap)`;
  const expressionAttributeValues = {
    ":emptyMap": {}
  };

  // Add each update to the expression
  Object.entries(updates).forEach(([field, value], index) => {
    updateExpression += `, ${mapName}.${field} = :val${index}`;
  });
}

```

```

        expressionAttributeValues[`:val${index}`] = value;
    });

    // Define the update parameters
    const params = {
        TableName: tableName,
        Key: key,
        UpdateExpression: updateExpression,
        ExpressionAttributeValues: expressionAttributeValues,
        ReturnValues: "UPDATED_NEW"
    };

    // Perform the update operation
    const response = await docClient.send(new UpdateCommand(params));

    return response;
}

/**
 * Get the current value of an item.
 *
 * Helper function to retrieve the current value of an item.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to get
 * @returns {Promise<Object|null>} - The item or null if not found
 */
async function getItem(
    config,
    tableName,
    key
) {
    // Initialize the DynamoDB client
    const client = new DynamoDBClient(config);
    const docClient = DynamoDBDocumentClient.from(client);

    // Define the get parameters
    const params = {
        TableName: tableName,
        Key: key
    };

    // Perform the get operation

```

```
const response = await docClient.send(new GetCommand(params));

// Return the item if it exists, otherwise null
return response.Item || null;
}

/**
 * Example of how to use the map attribute update functions.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "Users";
  const key = { UserId: "U12345" };

  console.log("Demonstrating different approaches to update map attributes in
  DynamoDB");

  try {
    // Example 1: Update a map attribute that might not exist (two-step approach)
    console.log("\nExample 1: Updating a map attribute that might not exist (two-
    step approach)");
    const response1 = await updateMapAttributeSafe(
      config,
      tableName,
      key,
      "Preferences",
      "Theme",
      "Dark"
    );

    console.log("Updated preferences:", response1.Attributes);

    // Example 2: Update a map attribute using if_not_exists (elegant approach)
    console.log("\nExample 2: Updating a map attribute using if_not_exists (elegant
    approach)");
    const response2 = await updateMapAttributeWithIfNotExists(
      config,
      tableName,
      key,
      "Settings",
      "NotificationsEnabled",
      true
    );
  }
}
```

```
console.log("Updated settings:", response2.Attributes);

// Example 3: Update a deeply nested attribute
console.log("\nExample 3: Updating a deeply nested attribute");
const response3 = await addToNestedMap(
  config,
  tableName,
  key,
  ["Profile", "Address", "City"],
  "Seattle"
);

console.log("Updated nested attribute:", response3.Attributes);

// Example 4: Update multiple fields in a map
console.log("\nExample 4: Updating multiple fields in a map");
const response4 = await updateMultipleMapFields(
  config,
  tableName,
  key,
  "ContactInfo",
  {
    Email: "user@example.com",
    Phone: "555-123-4567",
    PreferredContact: "Email"
  }
);

console.log("Updated multiple fields:", response4.Attributes);

// Get the final state of the item
console.log("\nFinal state of the item:");
const item = await getItem(config, tableName, key);
console.log(JSON.stringify(item, null, 2));

// Explain the benefits of different approaches
console.log("\nKey points about updating map attributes:");
console.log("1. Use if_not_exists to handle maps that might not exist");
console.log("2. Multiple updates can be combined in a single operation");
console.log("3. Deeply nested attributes require creating parent maps");
console.log("4. DynamoDB expressions are atomic - the entire update succeeds or fails");
```

```
    console.log("5. Using a single operation is more efficient than multiple
separate updates");

    } catch (error) {
        console.error("Error:", error);
    }
}

// Export the functions
module.exports = {
    updateMapAttributeSafe,
    updateMapAttributeWithIfNotExists,
    addToNestedMap,
    updateMultipleMapFields,
    getItem,
    exampleUsage
};

// Run the example if this file is executed directly
if (require.main === module) {
    exampleUsage();
}
```

- Einzelheiten zur API finden Sie [UpdateItem](#) in der AWS SDK für JavaScript API-Referenz.

Ausführen von Set-Operationen

Im folgenden Codebeispiel wird gezeigt, wie Set-Operationen in DynamoDB ausgeführt werden.

- Hinzufügen von Elementen zu einem Set-Attribut
- Entfernen von Elementen aus einem Set-Attribut
- Verwenden der Operationen ADD und DELETE mit Sätzen

SDK für JavaScript (v3)

Demonstrieren Sie die Bedienung von Sets mithilfe von AWS SDK für JavaScript.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
    DynamoDBDocumentClient,
```



```

    UpdateCommand,
    GetCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Add elements to a set attribute.
 *
 * This function demonstrates using the ADD operation to add elements to a set.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} setName - The name of the set attribute
 * @param {Array} values - The values to add to the set
 * @param {string} setType - The type of set ('string', 'number', or 'binary')
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function addToSet(
  config,
  tableName,
  key,
  setName,
  values,
  setType = 'string'
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Create the appropriate set type
  let setValues;
  if (setType === 'string') {
    setValues = new Set(values.map(String));
  } else if (setType === 'number') {
    setValues = new Set(values.map(Number));
  } else if (setType === 'binary') {
    setValues = new Set(values);
  } else {
    throw new Error(`Unsupported set type: ${setType}`);
  }

  // Define the update parameters using ADD
  const params = {
    TableName: tableName,

```

```

    Key: key,
    UpdateExpression: `ADD ${setName} :values`,
    ExpressionAttributeValues: {
      ":values": setValues
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return response;
}

/**
 * Remove elements from a set attribute.
 *
 * This function demonstrates using the DELETE operation to remove elements from a
 * set.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} setName - The name of the set attribute
 * @param {Array} values - The values to remove from the set
 * @param {string} setType - The type of set ('string', 'number', or 'binary')
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function removeFromSet(
  config,
  tableName,
  key,
  setName,
  values,
  setType = 'string'
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Create the appropriate set type
  let setValues;
  if (setType === 'string') {
    setValues = new Set(values.map(String));
  }

```

```

    } else if (setType === 'number') {
        setValues = new Set(values.map(Number));
    } else if (setType === 'binary') {
        setValues = new Set(values);
    } else {
        throw new Error(`Unsupported set type: ${setType}`);
    }

    // Define the update parameters using DELETE
    const params = {
        TableName: tableName,
        Key: key,
        UpdateExpression: `DELETE ${setName} :values`,
        ExpressionAttributeValues: {
            ":values": setValues
        },
        ReturnValues: "UPDATED_NEW"
    };

    // Perform the update operation
    const response = await docClient.send(new UpdateCommand(params));

    return response;
}

/**
 * Create a new set attribute with initial values.
 *
 * This function demonstrates using the SET operation to create a new set attribute.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} setName - The name of the set attribute
 * @param {Array} values - The initial values for the set
 * @param {string} setType - The type of set ('string', 'number', or 'binary')
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function createSet(
    config,
    tableName,
    key,
    setName,
    values,

```

```
    setType = 'string'
  ) {
    // Initialize the DynamoDB client
    const client = new DynamoDBClient(config);
    const docClient = DynamoDBDocumentClient.from(client);

    // Create the appropriate set type
    let setValues;
    if (setType === 'string') {
      setValues = new Set(values.map(String));
    } else if (setType === 'number') {
      setValues = new Set(values.map(Number));
    } else if (setType === 'binary') {
      setValues = new Set(values);
    } else {
      throw new Error(`Unsupported set type: ${setType}`);
    }

    // Define the update parameters using SET
    const params = {
      TableName: tableName,
      Key: key,
      UpdateExpression: `SET ${setName} = :values`,
      ExpressionAttributeValues: {
        ":values": setValues
      },
      ReturnValues: "UPDATED_NEW"
    };

    // Perform the update operation
    const response = await docClient.send(new UpdateCommand(params));

    return response;
  }

/**
 * Replace an entire set attribute with a new set of values.
 *
 * This function demonstrates using the SET operation to replace an entire set.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} setName - The name of the set attribute
```

```
* @param {Array} values - The new values for the set
* @param {string} setType - The type of set ('string', 'number', or 'binary')
* @returns {Promise<Object>} - The response from DynamoDB
*/
async function replaceSet(
  config,
  tableName,
  key,
  setName,
  values,
  setType = 'string'
) {
  // This is the same as createSet, but included for clarity of intent
  return await createSet(config, tableName, key, setName, values, setType);
}

/**
 * Remove the last element from a set and handle the empty set case.
 *
 * This function demonstrates what happens when you delete the last element of a
 * set.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} setName - The name of the set attribute
 * @returns {Promise<Object>} - The result of the operation
 */
async function removeLastElementFromSet(
  config,
  tableName,
  key,
  setName
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // First, get the current item to check the set
  const currentItem = await getItem(config, tableName, key);

  // Check if the set exists and has elements
  if (!currentItem || !currentItem[setName] || currentItem[setName].size === 0) {
    return {
```

```
        success: false,
        message: "Set doesn't exist or is already empty",
        item: currentItem
    };
}

// Get the set values
const setValues = Array.from(currentItem[setName]);

// If there's only one element left, remove the attribute entirely
if (setValues.length === 1) {
    // Define the update parameters to remove the attribute
    const params = {
        TableName: tableName,
        Key: key,
        UpdateExpression: `REMOVE ${setName}`,
        ReturnValues: "UPDATED_NEW"
    };

    // Perform the update operation
    await docClient.send(new UpdateCommand(params));

    return {
        success: true,
        message: "Last element removed, attribute has been deleted",
        removedValue: setValues[0]
    };
} else {
    // Otherwise, remove just the last element
    // Create a set with just the last element
    const lastElement = setValues[setValues.length - 1];
    const setType = typeof lastElement === 'number' ? 'number' : 'string';

    // Remove the last element
    const response = await removeFromSet(
        config,
        tableName,
        key,
        setName,
        [lastElement],
        setType
    );

    return {
```

```

        success: true,
        message: "Last element removed, set still contains elements",
        removedValue: lastElement,
        remainingSet: response.Attributes[setName]
    };
}
}

/**
 * Get the current value of an item.
 *
 * Helper function to retrieve the current value of an item.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to get
 * @returns {Promise<Object|null>} - The item or null if not found
 */
async function getItem(
    config,
    tableName,
    key
) {
    // Initialize the DynamoDB client
    const client = new DynamoDBClient(config);
    const docClient = DynamoDBDocumentClient.from(client);

    // Define the get parameters
    const params = {
        TableName: tableName,
        Key: key
    };

    // Perform the get operation
    const response = await docClient.send(new GetCommand(params));

    // Return the item if it exists, otherwise null
    return response.Item || null;
}

```

Beispiel für die Verwendung von Mengenoperationen mit AWS SDK für JavaScript.

```
/**
 * Example of how to work with sets in DynamoDB.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "Users";
  const key = { UserId: "U12345" };

  console.log("Demonstrating set operations in DynamoDB");

  try {
    // Example 1: Create a string set
    console.log("\nExample 1: Creating a string set");
    const response1 = await createSet(
      config,
      tableName,
      key,
      "Interests",
      ["Reading", "Hiking", "Cooking"],
      "string"
    );

    console.log("Created set:", response1.Attributes);

    // Example 2: Add elements to a set
    console.log("\nExample 2: Adding elements to a set");
    const response2 = await addToSet(
      config,
      tableName,
      key,
      "Interests",
      ["Photography", "Travel"],
      "string"
    );

    console.log("Updated set after adding elements:", response2.Attributes);

    // Example 3: Remove elements from a set
    console.log("\nExample 3: Removing elements from a set");
    const response3 = await removeFromSet(
      config,
      tableName,
```



```
    key,
    "Interests",
    ["Cooking"],
    "string"
  );

  console.log("Updated set after removing elements:", response3.Attributes);

  // Example 4: Create a number set
  console.log("\nExample 4: Creating a number set");
  const response4 = await createSet(
    config,
    tableName,
    key,
    "FavoriteNumbers",
    [7, 42, 99],
    "number"
  );

  console.log("Created number set:", response4.Attributes);

  // Example 5: Replace an entire set
  console.log("\nExample 5: Replacing an entire set");
  const response5 = await replaceSet(
    config,
    tableName,
    key,
    "Interests",
    ["Gaming", "Movies", "Music"],
    "string"
  );

  console.log("Replaced set:", response5.Attributes);

  // Example 6: Remove the last element from a set
  console.log("\nExample 6: Removing the last element from a set");

  // First, create a set with just one element
  await createSet(
    config,
    tableName,
    { UserId: "U67890" },
    "Tags",
    ["LastTag"],
```

```
    "string"
  );

  // Then, remove the last element
  const response6 = await removeLastElementFromSet(
    config,
    tableName,
    { UserId: "U67890" },
    "Tags"
  );

  console.log(response6.message);
  console.log("Removed value:", response6.removedValue);

  // Get the final state of the items
  console.log("\nFinal state of the items:");
  const item1 = await getItem(config, tableName, key);
  console.log("User U12345:", JSON.stringify(item1, null, 2));

  const item2 = await getItem(config, tableName, { UserId: "U67890" });
  console.log("User U67890:", JSON.stringify(item2, null, 2));

  // Explain set operations
  console.log("\nKey points about set operations in DynamoDB:");
  console.log("1. Use ADD to add elements to a set (duplicates are automatically removed)");
  console.log("2. Use DELETE to remove elements from a set");
  console.log("3. Use SET to create a new set or replace an existing one");
  console.log("4. DynamoDB supports three types of sets: string sets, number sets, and binary sets");
  console.log("5. When you delete the last element from a set, the attribute remains as an empty set");
  console.log("6. To remove an empty set, use the REMOVE operation");
  console.log("7. Sets automatically maintain unique values (no duplicates)");
  console.log("8. You cannot mix data types within a set");

} catch (error) {
  console.error("Error:", error);
}
}
```

- Einzelheiten zur API finden Sie [UpdateItem](#) in der AWS SDK für JavaScript API-Referenz.

Abfragen einer Tabelle mithilfe von Stapeln von PartiQL-Anweisungen

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Abrufen eines Stapels von Elementen mithilfe mehrerer SELECT-Anweisungen.
- Hinzufügen eines Stapels von Elementen hinzu, indem mehrere INSERT-Anweisungen ausgeführt werden.
- Aktualisieren eines Stapels von Elementen mithilfe mehrerer UPDATE-Anweisungen.
- Löschen eines Stapels von Elementen mithilfe mehrerer DELETE-Anweisungen.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Führen Sie Batch-PartiQL-Anweisungen aus.

```
import {
  BillingMode,
  CreateTableCommand,
  DeleteTableCommand,
  DescribeTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";
import { ScenarioInput } from "@aws-doc-sdk-examples/lib/scenario";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

const log = (msg) => console.log(`[SCENARIO] ${msg}`);
const tableName = "Cities";

export const main = async (confirmAll = false) => {
```

```

/**
 * Delete table if it exists.
 */
try {
  await client.send(new DescribeTableCommand({ TableName: tableName }));
  // If no error was thrown, the table exists.
  const input = new ScenarioInput(
    "deleteTable",
    `A table named ${tableName} already exists. If you choose not to delete
this table, the scenario cannot continue. Delete it?`,
    { type: "confirm", confirmAll },
  );
  const deleteTable = await input.handle({}, { confirmAll });
  if (deleteTable) {
    await client.send(new DeleteTableCommand({ tableName }));
  } else {
    console.warn(
      "Scenario could not run. Either delete ${tableName} or provide a unique
table name.",
    );
    return;
  }
} catch (caught) {
  if (
    caught instanceof Error &&
    caught.name === "ResourceNotFoundException"
  ) {
    // Do nothing. This means the table is not there.
  } else {
    throw caught;
  }
}

/**
 * Create a table.
 */

log("Creating a table.");
const createTableCommand = new CreateTableCommand({
  TableName: tableName,
  // This example performs a large write to the database.
  // Set the billing mode to PAY_PER_REQUEST to
  // avoid throttling the large write.
  BillingMode: BillingMode.PAY_PER_REQUEST,

```

```

// Define the attributes that are necessary for the key schema.
AttributeDefinitions: [
  {
    AttributeName: "name",
    // 'S' is a data type descriptor that represents a number type.
    // For a list of all data type descriptors, see the following link.
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
    AttributeType: "S",
  },
],
// The KeySchema defines the primary key. The primary key can be
// a partition key, or a combination of a partition key and a sort key.
// Key schema design is important. For more info, see
// https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/best-
practices.html
KeySchema: [{ AttributeName: "name", KeyType: "HASH" }],
});
await client.send(createTableCommand);
log(`Table created: ${tableName}.`);

/**
 * Wait until the table is active.
 */

// This polls with DescribeTableCommand until the requested table is 'ACTIVE'.
// You can't write to a table before it's active.
log("Waiting for the table to be active.");
await waitUntilTableExists({ client }, { TableName: tableName });
log("Table active.");

/**
 * Insert items.
 */

log("Inserting cities into the table.");
const addItemStatementCommand = new BatchExecuteStatementCommand({
  // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.insert.html
  Statements: [
    {
      Statement: `INSERT INTO ${tableName} value {'name':?, 'population':?}`,
      Parameters: ["Alachua", 10712],
    },
  ],

```

```

    {
      Statement: `INSERT INTO ${tableName} value {'name':?, 'population':?}`,
      Parameters: ["High Springs", 6415],
    },
  ],
});
await docClient.send(addItemsStatementCommand);
log("Cities inserted.");

/**
 * Select items.
 */

log("Selecting cities from the table.");
const selectItemsStatementCommand = new BatchExecuteStatementCommand({
  // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
  // reference.select.html
  Statements: [
    {
      Statement: `SELECT * FROM ${tableName} WHERE name=?`,
      Parameters: ["Alachua"],
    },
    {
      Statement: `SELECT * FROM ${tableName} WHERE name=?`,
      Parameters: ["High Springs"],
    },
  ],
});
const selectItemResponse = await docClient.send(selectItemsStatementCommand);
log(
  `Got cities: ${selectItemResponse.Responses.map(
    (r) => `${r.Item.name} (${r.Item.population})`,
  ).join(", ")}`
);

/**
 * Update items.
 */

log("Modifying the populations.");
const updateItemStatementCommand = new BatchExecuteStatementCommand({
  // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
  // reference.update.html
  Statements: [

```

```

    {
      Statement: `UPDATE ${tableName} SET population=? WHERE name=?`,
      Parameters: [10, "Alachua"],
    },
    {
      Statement: `UPDATE ${tableName} SET population=? WHERE name=?`,
      Parameters: [5, "High Springs"],
    },
  ],
});
await docClient.send(updateItemStatementCommand);
log("Updated cities.");

/**
 * Delete the items.
 */

log("Deleting the cities.");
const deleteItemStatementCommand = new BatchExecuteStatementCommand({
  // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.delete.html
  Statements: [
    {
      Statement: `DELETE FROM ${tableName} WHERE name=?`,
      Parameters: ["Alachua"],
    },
    {
      Statement: `DELETE FROM ${tableName} WHERE name=?`,
      Parameters: ["High Springs"],
    },
  ],
});
await docClient.send(deleteItemStatementCommand);
log("Cities deleted.");

/**
 * Delete the table.
 */

log("Deleting the table.");
const deleteTableCommand = new DeleteTableCommand({ TableName: tableName });
await client.send(deleteTableCommand);
log("Table deleted.");
};

```

- Einzelheiten zur API finden Sie [BatchExecuteStatement](#) in der AWS SDK für JavaScript API-Referenz.

Abfragen einer Tabelle mit PartiQL

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Abrufen eines Elementes durch Ausführen einer SELECT-Anweisung.
- Hinzufügen eines Elementes durch Ausführung einer INSERT-Anweisung.
- Aktualisieren eines Elementes durch Ausführung einer UPDATE-Anweisung.
- Löschen eines Elementes durch Ausführung einer DELETE-Anweisung.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Führen Sie einzelne PartiQL-Anweisungen aus.

```
import {
  BillingMode,
  CreateTableCommand,
  DeleteTableCommand,
  DescribeTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  ExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";
import { ScenarioInput } from "@aws-doc-sdk-examples/lib/scenario";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);
```



```

const log = (msg) => console.log(`[SCENARIO] ${msg}`);
const tableName = "SingleOriginCoffees";

export const main = async (confirmAll = false) => {
  /**
   * Delete table if it exists.
   */
  try {
    await client.send(new DescribeTableCommand({ TableName: tableName }));
    // If no error was thrown, the table exists.
    const input = new ScenarioInput(
      "deleteTable",
      `A table named ${tableName} already exists. If you choose not to delete
this table, the scenario cannot continue. Delete it?`,
      { type: "confirm", confirmAll },
    );
    const deleteTable = await input.handle({});
    if (deleteTable) {
      await client.send(new DeleteTableCommand({ tableName }));
    } else {
      console.warn(
        "Scenario could not run. Either delete ${tableName} or provide a unique
table name.",
      );
      return;
    }
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "ResourceNotFoundException"
    ) {
      // Do nothing. This means the table is not there.
    } else {
      throw caught;
    }
  }

  /**
   * Create a table.
   */

  log("Creating a table.");
  const createTableCommand = new CreateTableCommand({

```

```

    TableName: tableName,
    // This example performs a large write to the database.
    // Set the billing mode to PAY_PER_REQUEST to
    // avoid throttling the large write.
    BillingMode: BillingMode.PAY_PER_REQUEST,
    // Define the attributes that are necessary for the key schema.
    AttributeDefinitions: [
      {
        AttributeName: "varietal",
        // 'S' is a data type descriptor that represents a number type.
        // For a list of all data type descriptors, see the following link.
        // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/
Programming.LowLevelAPI.html#Programming.LowLevelAPI.DataTypeDescriptors
        AttributeType: "S",
      },
    ],
    // The KeySchema defines the primary key. The primary key can be
    // a partition key, or a combination of a partition key and a sort key.
    // Key schema design is important. For more info, see
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/best-
practices.html
    KeySchema: [{ AttributeName: "varietal", KeyType: "HASH" }],
  });
  await client.send(createTableCommand);
  log(`Table created: ${tableName}.`);

  /**
   * Wait until the table is active.
   */

  // This polls with DescribeTableCommand until the requested table is 'ACTIVE'.
  // You can't write to a table before it's active.
  log("Waiting for the table to be active.");
  await waitUntilTableExists({ client }, { TableName: tableName });
  log("Table active.");

  /**
   * Insert an item.
   */

  log("Inserting a coffee into the table.");
  const addItemStatementCommand = new ExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.insert.html

```

```
    Statement: `INSERT INTO ${tableName} value {'varietal':?, 'profile':?}`,
    Parameters: ["arabica", ["chocolate", "floral"]],
  });
  await client.send(addItemStatementCommand);
  log("Coffee inserted.");

  /**
   * Select an item.
   */

  log("Selecting the coffee from the table.");
  const selectItemStatementCommand = new ExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.select.html
    Statement: `SELECT * FROM ${tableName} WHERE varietal=?`,
    Parameters: ["arabica"],
  });
  const selectItemResponse = await docClient.send(selectItemStatementCommand);
  log(`Got coffee: ${JSON.stringify(selectItemResponse.Items[0])}`);

  /**
   * Update the item.
   */

  log("Add a flavor profile to the coffee.");
  const updateItemStatementCommand = new ExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.update.html
    Statement: `UPDATE ${tableName} SET profile=list_append(profile, ?) WHERE
varietal=?`,
    Parameters: [["fruity"], "arabica"],
  });
  await client.send(updateItemStatementCommand);
  log("Updated coffee");

  /**
   * Delete the item.
   */

  log("Deleting the coffee.");
  const deleteItemStatementCommand = new ExecuteStatementCommand({
    // https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ql-
reference.delete.html
    Statement: `DELETE FROM ${tableName} WHERE varietal=?`,
```

```

    Parameters: ["arabica"],
  });
  await docClient.send(deleteItemStatementCommand);
  log("Coffee deleted.");

  /**
   * Delete the table.
   */

  log("Deleting the table.");
  const deleteTableCommand = new DeleteTableCommand({ TableName: tableName });
  await client.send(deleteTableCommand);
  log("Table deleted.");
};

```

- Einzelheiten zur API finden Sie [ExecuteStatement](#) in der AWS SDK für JavaScript API-Referenz.

Abfragen einer Tabelle mit einem globalen sekundären Index

Im folgenden Codebeispiel wird gezeigt, wie eine Tabelle mit einem globalen sekundären Index abgefragt wird.

- Abfragen einer DynamoDB-Tabelle mit ihrem Primärschlüssel
- Abfragen eines globalen sekundären Index (GSI) für alternative Zugriffsmuster
- Vergleichen von Tabellenabfragen und GSI-Abfragen

SDK für JavaScript (v3)

Fragen Sie eine DynamoDB-Tabelle mit dem Primärschlüssel mit ab. AWS SDK für JavaScript

```

const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table using the primary key
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} userId - The user ID to query by (partition key)
 * @returns {Promise<Object>} - The query response
 */

```

```

async function queryTable(
  config,
  tableName,
  userId
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input for the base table
    const input = {
      TableName: tableName,
      KeyConditionExpression: "user_id = :userId",
      ExpressionAttributeValues: {
        ":userId": { S: userId }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying table: ${error}`);
    throw error;
  }
}

```

Fragen Sie einen DynamoDB Global Secondary Index (GSI) mit ab. AWS SDK für JavaScript

```

/**
 * Queries a DynamoDB Global Secondary Index (GSI)
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} indexName - The name of the GSI to query
 * @param {string} gameId - The game ID to query by (GSI partition key)
 * @returns {Promise<Object>} - The query response
 */
async function queryGSI(
  config,
  tableName,
  indexName,

```

```
    gameId
  ) {
    try {
      // Create DynamoDB client
      const client = new DynamoDBClient(config);

      // Construct the query input for the GSI
      const input = {
        TableName: tableName,
        IndexName: indexName,
        KeyConditionExpression: "game_id = :gameId",
        ExpressionAttributeValues: {
          ":gameId": { S: gameId }
        }
      };

      // Execute the query
      const command = new QueryCommand(input);
      return await client.send(command);
    } catch (error) {
      console.error(`Error querying GSI: ${error}`);
      throw error;
    }
  }
}
```

- Weitere API-Informationen finden Sie unter [Abfragen](#) in der AWS SDK für JavaScript -API-Referenz.

Abfragen einer Tabelle mithilfe einer „begins_with“-Bedingung

Im folgenden Codebeispiel wird gezeigt, wie eine Tabelle mit einer „begins_with“-Bedingung abgefragt wird.

- Verwenden der Funktion „begins_with“ in einem Schlüsselbedingungsausdruck
- Filtern der Elemente auf Grundlage eines Präfixmusters für den Sortierschlüssel

SDK für JavaScript (v3)

Fragen Sie mit AWS SDK für JavaScript eine DynamoDB-Tabelle mithilfe einer „begins_with“-Bedingung im Sortierschlüssel ab.

```

const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table for items where the sort key begins with a specific
 * prefix
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {string} sortKeyName - The name of the sort key
 * @param {string} prefix - The prefix to match at the beginning of the sort key
 * @returns {Promise<Object>} - The query response
 */
async function queryWithBeginsWith(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  sortKeyName,
  prefix
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "#pk = :pkValue AND begins_with(#sk, :prefix)",
      ExpressionAttributeNames: {
        "#pk": partitionKeyName,
        "#sk": sortKeyName
      },
      ExpressionAttributeValues: {
        ":pkValue": { S: partitionKeyValue },
        ":prefix": { S: prefix }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  }
}

```

```

    } catch (error) {
      console.error(`Error querying with begins_with: ${error}`);
      throw error;
    }
  }
}

```

- Weitere API-Informationen finden Sie unter [Abfragen](#) in der AWS SDK für JavaScript -API-Referenz.

Abfragen einer Tabelle mit einem Datumsbereich

Im folgenden Codebeispiel wird gezeigt, wie eine Tabelle mithilfe eines Datumsbereichs im Sortierschlüssel abgefragt wird.

- Abfragen von Elementen innerhalb eines bestimmten Datumsbereichs
- Verwenden von Vergleichsoperatoren für Sortierschlüssel im Datumsformat

SDK für JavaScript (v3)

Fragen Sie eine DynamoDB-Tabelle nach Elementen innerhalb eines Datumsbereichs mit ab.
AWS SDK für JavaScript

```

const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table for items within a specific date range on the sort key
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {string} sortKeyName - The name of the sort key (must be a date/time
attribute)
 * @param {Date} startDate - The start date for the range query
 * @param {Date} endDate - The end date for the range query
 * @returns {Promise<Object>} - The query response
 */
async function queryByDateRangeOnSortKey(
  config,
  tableName,

```



```
partitionKeyName,
partitionKeyValue,
sortKeyName,
startDate,
endDate
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Format dates as ISO strings for DynamoDB
    const formattedStartDate = startDate.toISOString();
    const formattedEndDate = endDate.toISOString();

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: '#pk = :pkValue AND #sk BETWEEN :startDate
AND :endDate',
      ExpressionAttributeNames: {
        '#pk': partitionKeyName,
        '#sk': sortKeyName
      },
      ExpressionAttributeValues: {
        ':pkValue': { S: partitionKeyValue },
        ':startDate': { S: formattedStartDate },
        ':endDate': { S: formattedEndDate }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying by date range on sort key: ${error}`);
    throw error;
  }
}
```

- Weitere API-Informationen finden Sie unter [Abfragen](#) in der AWS SDK für JavaScript -API-Referenz.

Abfragen einer Tabelle mit einem komplexen Filterausdruck

Im folgenden Codebeispiel wird gezeigt, wie eine Tabelle mit einem komplexen Filterausdruck abgefragt wird.

- Anwenden von komplexen Filterausdrücken auf Abfrageergebnisse
- Kombinieren von mehreren Bedingungen über logische Operatoren
- Filtern von Elementen auf der Grundlage von Nicht-Schlüsselattributen.

SDK für JavaScript (v3)

Fragen Sie eine DynamoDB-Tabelle mit einem komplexen Filterausdruck ab mit. AWS SDK für JavaScript

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table with a complex filter expression
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {number|string} minViews - Minimum number of views for filtering
 * @param {number|string} minReplies - Minimum number of replies for filtering
 * @param {string} requiredTag - Tag that must be present in the item's tags set
 * @returns {Promise<Object>} - The query response
 */
async function queryWithComplexFilter(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  minViews,
  minReplies,
  requiredTag
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input
```

```
const input = {
  TableName: tableName,
  KeyConditionExpression: "#pk = :pkValue",
  FilterExpression: "views >= :minViews AND replies >= :minReplies AND
contains(tags, :tag)",
  ExpressionAttributeNames: {
    "#pk": partitionKeyName
  },
  ExpressionAttributeValues: {
    ":pkValue": { S: partitionKeyValue },
    ":minViews": { N: minViews.toString() },
    ":minReplies": { N: minReplies.toString() },
    ":tag": { S: requiredTag }
  }
};

// Execute the query
const command = new QueryCommand(input);
return await client.send(command);
} catch (error) {
  console.error(`Error querying with complex filter: ${error}`);
  throw error;
}
}
```

- Weitere API-Informationen finden Sie unter [Abfragen](#) in der AWS SDK für JavaScript -API-Referenz.

Abfragen einer Tabelle mit einem dynamischen Filterausdruck

Im folgenden Codebeispiel wird gezeigt, wie eine Tabelle mit einem dynamischen Filterausdruck abgefragt wird.

- Erstellen von Filterausdrücken dynamisch zur Laufzeit
- Konstruieren von Filterbedingungen auf der Grundlage von Benutzereingaben oder Anwendungsstatus
- Hinzufügen oder Entfernen von Filterkriterien unter bestimmten Bedingungen

SDK für JavaScript (v3)

Fragen Sie eine DynamoDB-Tabelle mit einem dynamisch erstellten Filterausdruck ab mit. AWS SDK für JavaScript

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

async function queryWithDynamicFilter(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  sortKeyName,
  sortKeyValue,
  filterParams = {}
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Initialize filter expression components
    let filterExpressions = [];
    const expressionAttributeValues = {
      ":pkValue": { S: partitionKeyValue },
      ":skValue": { S: sortKeyValue }
    };
    const expressionAttributeNames = {
      "#pk": partitionKeyName,
      "#sk": sortKeyName
    };

    // Add status filter if provided
    if (filterParams.status) {
      filterExpressions.push("status = :status");
      expressionAttributeValues[":status"] = { S: filterParams.status };
    }

    // Add minimum views filter if provided
    if (filterParams.minViews !== undefined) {
      filterExpressions.push("views >= :minViews");
      expressionAttributeValues[":minViews"] = { N:
filterParams.minViews.toString() };
    }
  }
}
```

```
// Add author filter if provided
if (filterParams.author) {
  filterExpressions.push("author = :author");
  expressionAttributeValues[":author"] = { S: filterParams.author };
}

// Construct the query input
const input = {
  TableName: tableName,
  KeyConditionExpression: "#pk = :pkValue AND #sk = :skValue"
};

// Add filter expression if any filters were provided
if (filterExpressions.length > 0) {
  input.FilterExpression = filterExpressions.join(" AND ");
}

// Add expression attribute names and values
input.ExpressionAttributeNames = expressionAttributeNames;
input.ExpressionAttributeValues = expressionAttributeValues;

// Execute the query
const command = new QueryCommand(input);
return await client.send(command);
} catch (error) {
  console.error(`Error querying with dynamic filter: ${error}`);
  throw error;
}
}
```

- Weitere API-Informationen finden Sie unter [Abfragen](#) in der AWS SDK für JavaScript -API-Referenz.

Abfragen einer Tabelle mit verschachtelten Attributen

Im folgenden Codebeispiel wird gezeigt, wie eine Tabelle mit verschachtelten Attributen abgefragt wird.

- Zugreifen und Filtern nach verschachtelten Attributen in DynamoDB-Elementen
- Verwenden von Dokumentpfadausdrücken, um auf verschachtelte Elemente zu verweisen

SDK für JavaScript (v3)

Fragen Sie eine DynamoDB-Tabelle mit verschachtelten Attributen ab mit. AWS SDK für JavaScript

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table filtering on a nested attribute
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} productId - The product ID to query by (partition key)
 * @param {string} category - The category to filter by (nested attribute)
 * @returns {Promise<Object>} - The query response
 */
async function queryWithNestedAttribute(
  config,
  tableName,
  productId,
  category
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: "product_id = :productId",
      FilterExpression: "details.category = :category",
      ExpressionAttributeValues: {
        ":productId": { S: productId },
        ":category": { S: category }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying with nested attribute: ${error}`);
    throw error;
  }
}
```

```
}

```

- Weitere API-Informationen finden Sie unter [Abfragen](#) in der AWS SDK für JavaScript -API-Referenz.

Abfragen einer Tabelle mit Paginierung

Das folgende Codebeispiel zeigt, wie eine Tabelle mit Paginierung abgefragt wird.

- Implementieren der Paginierung für DynamoDB-Abfrageergebnisse
- Verwenden Sie die `LastEvaluatedKey`, um nachfolgende Seiten abzurufen.
- Steuern der Anzahl der Elemente pro Seite mit dem Parameter „Limit“

SDK für JavaScript (v3)

Fragen Sie eine DynamoDB-Tabelle mit Paginierung ab mit. AWS SDK für JavaScript

```
/**
 * Example demonstrating how to handle large query result sets in DynamoDB using
 * pagination
 *
 * This example shows:
 * - How to use pagination to handle large result sets
 * - How to use LastEvaluatedKey to retrieve the next page of results
 * - How to construct subsequent query requests using ExclusiveStartKey
 */
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table with pagination to handle large result sets
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {number} pageSize - Number of items per page
 * @returns {Promise<Array>} - All items from the query
 */
async function queryWithPagination(
```

```
    config,
    tableName,
    partitionKeyName,
    partitionKeyValue,
    pageSize = 25
  ) {
    try {
      // Create DynamoDB client
      const client = new DynamoDBClient(config);

      // Initialize variables for pagination
      let lastEvaluatedKey = undefined;
      const allItems = [];
      let pageCount = 0;

      // Loop until all pages are retrieved
      do {
        // Construct the query input
        const input = {
          TableName: tableName,
          KeyConditionExpression: "#pk = :pkValue",
          Limit: pageSize,
          ExpressionAttributeNames: {
            "#pk": partitionKeyName
          },
          ExpressionAttributeValues: {
            ":pkValue": { S: partitionKeyValue }
          }
        };

        // Add ExclusiveStartKey if we have a LastEvaluatedKey from a previous query
        if (lastEvaluatedKey) {
          input.ExclusiveStartKey = lastEvaluatedKey;
        }

        // Execute the query
        const command = new QueryCommand(input);
        const response = await client.send(command);

        // Process the current page of results
        pageCount++;
        console.log(`Processing page ${pageCount} with ${response.Items.length}
items`);
      } while (response.LastEvaluatedKey);
    } catch (err) {
      console.error('Error: ', err);
    }
  }
}
```



```

    // Add the items from this page to our collection
    if (response.Items && response.Items.length > 0) {
        allItems.push(...response.Items);
    }

    // Get the LastEvaluatedKey for the next page
    lastEvaluatedKey = response.LastEvaluatedKey;

} while (lastEvaluatedKey); // Continue until there are no more pages

console.log(`Query complete. Retrieved ${allItems.length} items in ${pageCount}
pages.`);
return allItems;
} catch (error) {
    console.error(`Error querying with pagination: ${error}`);
    throw error;
}
}

/**
 * Example usage:
 *
 * // Query all items in the "AWS DynamoDB" forum with pagination
 * const allItems = await queryWithPagination(
 *   { region: "us-west-2" },
 *   "ForumThreads",
 *   "ForumName",
 *   "AWS DynamoDB",
 *   25 // 25 items per page
 * );
 *
 * console.log(`Total items retrieved: ${allItems.length}`);
 *
 * // Notes on pagination:
 * // - LastEvaluatedKey contains the primary key of the last evaluated item
 * // - When LastEvaluatedKey is undefined/null, there are no more items to retrieve
 * // - ExclusiveStartKey tells DynamoDB where to start the next page
 * // - Pagination helps manage memory usage for large result sets
 * // - Each page requires a separate network request to DynamoDB
 */

module.exports = { queryWithPagination };

```

- Weitere API-Informationen finden Sie unter [Abfragen](#) in der AWS SDK für JavaScript -API-Referenz.

Abfragen einer Tabelle mit strikt konsistenten Lesevorgängen

Im folgenden Codebeispiel wird gezeigt, wie eine Tabelle mit strikt konsistenten Lesevorgängen abgefragt wird.

- Konfigurieren des Konsistenzniveaus für DynamoDB-Abfragen
- Verwenden Sie stark konsistente Lesevorgänge, um die meisten Daten zu erhalten. up-to-date
- Erkennen der Kompromisse zwischen letztendlicher Konsistenz und starker Konsistenz

SDK für JavaScript (v3)

Fragen Sie eine DynamoDB-Tabelle mit konfigurierbarer Lesekonsistenz ab mit. AWS SDK für JavaScript

```
const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table with configurable read consistency
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {boolean} useConsistentRead - Whether to use strongly consistent reads
 * @returns {Promise<Object>} - The query response
 */
async function queryWithConsistentRead(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  useConsistentRead = false
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Construct the query input
```

```

const input = {
  TableName: tableName,
  KeyConditionExpression: "#pk = :pkValue",
  ExpressionAttributeNames: {
    "#pk": partitionKeyName
  },
  ExpressionAttributeValues: {
    ":pkValue": { S: partitionKeyValue }
  },
  ConsistentRead: useConsistentRead
};

// Execute the query
const command = new QueryCommand(input);
return await client.send(command);
} catch (error) {
  console.error(`Error querying with consistent read: ${error}`);
  throw error;
}
}

```

- Weitere API-Informationen finden Sie unter [Abfragen](#) in der AWS SDK für JavaScript -API-Referenz.

Abfragen von Daten mit PartiQL SELECT

Im folgenden Codebeispiel wird gezeigt, wie Sie Daten mit PartiQL-SELECT-Anweisungen abfragen.

SDK für JavaScript (v3)

Fragen Sie Elemente aus einer DynamoDB-Tabelle mithilfe von PartiQL SELECT-Anweisungen mit ab. AWS SDK für JavaScript

```

/**
 * This example demonstrates how to query items from a DynamoDB table using PartiQL.
 * It shows different ways to select data with various index types.
 */
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  ExecuteStatementCommand,

```

```

    BatchExecuteStatementCommand,
  } from "@aws-sdk/lib-dynamodb";

/**
 * Select all items from a DynamoDB table using PartiQL.
 * Note: This should be used with caution on large tables.
 *
 * @param tableName - The name of the DynamoDB table
 * @returns The response from the ExecuteStatementCommand
 */
export const selectAllItems = async (tableName: string) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `SELECT * FROM "${tableName}"`,
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Items retrieved successfully");
    return data;
  } catch (err) {
    console.error("Error retrieving items:", err);
    throw err;
  }
};

/**
 * Select an item by its primary key using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @returns The response from the ExecuteStatementCommand
 */
export const selectItemByPartitionKey = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

```

```

const params = {
  Statement: `SELECT * FROM "${tableName}" WHERE ${partitionKeyName} = ?`,
  Parameters: [partitionKeyValue],
};

try {
  const data = await docClient.send(new ExecuteStatementCommand(params));
  console.log("Item retrieved successfully");
  return data;
} catch (err) {
  console.error("Error retrieving item:", err);
  throw err;
}
};

/**
 * Select an item by its composite key (partition key + sort key) using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @param sortKeyName - The name of the sort key attribute
 * @param sortKeyValue - The value of the sort key
 * @returns The response from the ExecuteStatementCommand
 */
export const selectItemByCompositeKey = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number,
  sortKeyName: string,
  sortKeyValue: string | number
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `SELECT * FROM "${tableName}" WHERE ${partitionKeyName} = ? AND
${sortKeyName} = ?`,
    Parameters: [partitionKeyValue, sortKeyValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item retrieved successfully");
  }
};

```

```
    return data;
  } catch (err) {
    console.error("Error retrieving item:", err);
    throw err;
  }
};

/**
 * Select items using a filter condition with PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param filterAttribute - The attribute to filter on
 * @param filterValue - The value to filter by
 * @returns The response from the ExecuteStatementCommand
 */
export const selectItemsWithFilter = async (
  tableName: string,
  filterAttribute: string,
  filterValue: string | number
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `SELECT * FROM "${tableName}" WHERE ${filterAttribute} = ?`,
    Parameters: [filterValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Items retrieved successfully");
    return data;
  } catch (err) {
    console.error("Error retrieving items:", err);
    throw err;
  }
};

/**
 * Select items using a begins_with function for prefix matching.
 * This is useful for querying hierarchical data.
 *
 * @param tableName - The name of the DynamoDB table
 * @param attributeName - The attribute to check for prefix

```

```

* @param prefix - The prefix to match
* @returns The response from the ExecuteStatementCommand
*/
export const selectItemsByPrefix = async (
  tableName: string,
  attributeName: string,
  prefix: string
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `SELECT * FROM "${tableName}" WHERE
begins_with(${attributeName}, ?)`,
    Parameters: [prefix],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Items retrieved successfully");
    return data;
  } catch (err) {
    console.error("Error retrieving items:", err);
    throw err;
  }
};

/**
* Select items using a between condition for range queries.
*
* @param tableName - The name of the DynamoDB table
* @param attributeName - The attribute to check for range
* @param startValue - The start value of the range
* @param endValue - The end value of the range
* @returns The response from the ExecuteStatementCommand
*/
export const selectItemsByRange = async (
  tableName: string,
  attributeName: string,
  startValue: number | string,
  endValue: number | string
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

```

```
const params = {
  Statement: `SELECT * FROM "${tableName}" WHERE ${attributeName} BETWEEN ? AND ?`,
  Parameters: [startValue, endValue],
};

try {
  const data = await docClient.send(new ExecuteStatementCommand(params));
  console.log("Items retrieved successfully");
  return data;
} catch (err) {
  console.error("Error retrieving items:", err);
  throw err;
}
};

/**
 * Example usage showing how to select items with different index types
 */
export const selectExamples = async () => {
  // Select all items from a table (use with caution on large tables)
  await selectAllItems("UsersTable");

  // Select by partition key (simple primary key)
  await selectItemByPartitionKey("UsersTable", "userId", "user123");

  // Select by composite key (partition key + sort key)
  await selectItemByCompositeKey("OrdersTable", "orderId", "order456", "productId", "prod789");

  // Select with a filter condition (can use any attribute)
  await selectItemsWithFilter("UsersTable", "userType", "premium");

  // Select items with a prefix (useful for hierarchical data)
  await selectItemsByPrefix("ProductsTable", "category", "electronics");

  // Select items within a range (useful for numeric or date ranges)
  await selectItemsByRange("OrdersTable", "orderDate", "2023-01-01", "2023-12-31");
};
```


- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für JavaScript - API-Referenz.
 - [BatchExecuteStatement](#)
 - [ExecuteStatement](#)

Abfragen nach TTL-Elementen

Im folgenden Codebeispiel wird gezeigt, wie TTL-Elemente abgefragt werden.

SDK für JavaScript (v3)

Abfragen eines gefilterten Ausdrucks zum Sammeln von TTL-Elementen in einer DynamoDB-Tabelle mithilfe von AWS SDK für JavaScript

```
import { DynamoDBClient, QueryCommand } from "@aws-sdk/client-dynamodb";
import { marshall, unmarshall } from "@aws-sdk/util-dynamodb";

export const queryFiltered = async (tableName, primaryKey, region = 'us-east-1') =>
{
  const client = new DynamoDBClient({
    region: region,
    endpoint: `https://dynamodb.${region}.amazonaws.com`
  });

  const currentTime = Math.floor(Date.now() / 1000);

  const params = {
    TableName: tableName,
    KeyConditionExpression: "#pk = :pk",
    FilterExpression: "#ea > :ea",
    ExpressionAttributeNames: {
      "#pk": "primaryKey",
      "#ea": "expireAt"
    },
    ExpressionAttributeValues: marshall({
      ":pk": primaryKey,
      ":ea": currentTime
    })
  };

  try {
    const { Items } = await client.send(new QueryCommand(params));
```

```

        Items.forEach(item => {
            console.log(unmarshall(item))
        });
        return Items;
    } catch (err) {
        console.error(`Error querying items: ${err}`);
        throw err;
    }
}

// Example usage (commented out for testing)
// queryFiltered('your-table-name', 'your-partition-key-value');
```

- Weitere API-Informationen finden Sie unter [Abfragen](#) in der AWS SDK für JavaScript -API-Referenz.

Abfragen von Tabellen mithilfe von Datums- und Uhrzeitmustern

Im folgenden Codebeispiel wird gezeigt, wie Datums- und Uhrzeitmuster zum Abfragen von Tabellen verwendet werden.

- date/time Werte in DynamoDB speichern und abfragen.
- Implementieren von Datumsbereichsabfragen mithilfe von Sortierschlüsseln
- Formatieren von Datumszeichenfolgen für effektive Abfragen

SDK für JavaScript (v3)

Abfrage unter Verwendung von Datumsbereichen in Sortierschlüsseln mit AWS SDK für JavaScript.

```

const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table for items within a specific date range on the sort key
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
```

```

* @param {string} sortKeyName - The name of the sort key (must be a date/time
attribute)
* @param {Date} startDate - The start date for the range query
* @param {Date} endDate - The end date for the range query
* @returns {Promise<Object>} - The query response
*/
async function queryByDateRangeOnSortKey(
  config,
  tableName,
  partitionKeyName,
  partitionKeyValue,
  sortKeyName,
  startDate,
  endDate
) {
  try {
    // Create DynamoDB client
    const client = new DynamoDBClient(config);

    // Format dates as ISO strings for DynamoDB
    const formattedStartDate = startDate.toISOString();
    const formattedEndDate = endDate.toISOString();

    // Construct the query input
    const input = {
      TableName: tableName,
      KeyConditionExpression: '#pk = :pkValue AND #sk BETWEEN :startDate
AND :endDate',
      ExpressionAttributeNames: {
        '#pk': partitionKeyName,
        '#sk': sortKeyName
      },
      ExpressionAttributeValues: {
        ':pkValue': { S: partitionKeyValue },
        ':startDate': { S: formattedStartDate },
        ':endDate': { S: formattedEndDate }
      }
    };

    // Execute the query
    const command = new QueryCommand(input);
    return await client.send(command);
  } catch (error) {
    console.error(`Error querying by date range on sort key: ${error}`);
  }
}

```

```

        throw error;
    }
}

```

Abfrage mit Datums- und Uhrzeitvariablen mit AWS SDK für JavaScript

```

const { DynamoDBClient, QueryCommand } = require("@aws-sdk/client-dynamodb");

/**
 * Queries a DynamoDB table for items within a specific date range
 *
 * @param {Object} config - AWS SDK configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key
 * @param {string} partitionKeyValue - The value of the partition key
 * @param {string} dateKeyName - The name of the date attribute to filter on
 * @param {Date} startDate - The start date for the range query
 * @param {Date} endDate - The end date for the range query
 * @returns {Promise<Object>} - The query response
 */
async function queryByDateRange(
    config,
    tableName,
    partitionKeyName,
    partitionKeyValue,
    dateKeyName,
    startDate,
    endDate
) {
    try {
        // Create DynamoDB client
        const client = new DynamoDBClient(config);

        // Format dates as ISO strings for DynamoDB
        const formattedStartDate = startDate.toISOString();
        const formattedEndDate = endDate.toISOString();

        // Construct the query input
        const input = {
            TableName: tableName,
            KeyConditionExpression: `#pk = :pkValue AND #dateAttr BETWEEN :startDate
AND :endDate`,

```

```
    ExpressionAttributeNames: {
      "#pk": partitionKeyName,
      "#dateAttr": dateKeyName
    },
    ExpressionAttributeValues: {
      ":pkValue": { S: partitionKeyValue },
      ":startDate": { S: formattedStartDate },
      ":endDate": { S: formattedEndDate }
    }
  };

  // Execute the query
  const command = new QueryCommand(input);
  return await client.send(command);
} catch (error) {
  console.error(`Error querying by date range: ${error}`);
  throw error;
}
```

- Weitere API-Informationen finden Sie unter [Abfragen](#) in der AWS SDK für JavaScript -API-Referenz.

Verstehen der Reihenfolge der Aktualisierungsausdrücke

Im folgenden Codebeispiel wird die Reihenfolge der Aktualisierungsausdrücke erklärt.

- Erfahren, wie DynamoDB Aktualisierungsausdrücke verarbeitet
- Verstehen der Reihenfolge der Operationen in Aktualisierungsausdrücken
- Vermeiden von unerwarteten Ergebnissen durch Verständnis der Ausdrucksauswertung

SDK für JavaScript (v3)

Demonstrieren Sie die Reihenfolge der Aktualisierungsausdrücke mit AWS SDK für JavaScript.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  GetCommand,
```

```
PutCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Update an item with multiple actions in a single update expression.
 *
 * This function demonstrates how to use multiple actions in a single update
 * expression
 * and how DynamoDB processes these actions.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The primary key of the item to update
 * @param {string} updateExpression - The update expression with multiple actions
 * @param {Object} [expressionAttributeNames] - Expression attribute name
 * placeholders
 * @param {Object} [expressionAttributeValues] - Expression attribute value
 * placeholders
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateWithMultipleActions(
  config,
  tableName,
  key,
  updateExpression,
  expressionAttributeNames,
  expressionAttributeValues
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Prepare the update parameters
  const updateParams = {
    TableName: tableName,
    Key: key,
    UpdateExpression: updateExpression,
    ReturnValues: "UPDATED_NEW"
  };

  // Add expression attribute names if provided
  if (expressionAttributeNames) {
    updateParams.ExpressionAttributeNames = expressionAttributeNames;
  }
}
```

```
// Add expression attribute values if provided
if (expressionAttributeValues) {
  updateParams.ExpressionAttributeValues = expressionAttributeValues;
}

// Execute the update
const response = await docClient.send(new UpdateCommand(updateParams));

return response;
}

/**
 * Demonstrate that variables hold copies of existing values before modifications.
 *
 * This function creates an item with initial values, then updates it with an
 * expression
 * that uses the values of attributes before they are modified in the same
 * expression.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The primary key of the item to create and update
 * @returns {Promise<Object>} - A dictionary containing the results of the
 * demonstration
 */
async function demonstrateValueCopying(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Step 1: Create an item with initial values
  const initialItem = { ...key, a: 1, b: 2, c: 3 };

  await docClient.send(new PutCommand({
    TableName: tableName,
    Item: initialItem
  }));

  // Step 2: Get the item to verify initial state
```

```

const responseBefore = await docClient.send(new GetCommand({
  TableName: tableName,
  Key: key
}));

const itemBefore = responseBefore.Item || {};

// Step 3: Update the item with an expression that uses values before they are
// modified
// This expression removes 'a', then sets 'b' to the value of 'a', and 'c' to the
// value of 'b'
const updateResponse = await docClient.send(new UpdateCommand({
  TableName: tableName,
  Key: key,
  UpdateExpression: "REMOVE a SET b = a, c = b",
  ReturnValues: "UPDATED_NEW"
}));

// Step 4: Get the item to verify final state
const responseAfter = await docClient.send(new GetCommand({
  TableName: tableName,
  Key: key
}));

const itemAfter = responseAfter.Item || {};

// Return the results
return {
  initialState: itemBefore,
  updateResponse: updateResponse,
  finalState: itemAfter
};
}

/**
 * Demonstrate the order in which different action types are processed.
 *
 * This function creates an item with initial values, then updates it with an
 * expression
 * that includes multiple action types (SET, REMOVE, ADD, DELETE) to show the order
 * in which they are processed.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table

```



```
* @param {Object} key - The primary key of the item to create and update
* @returns {Promise<Object>} - A dictionary containing the results of the
demonstration
*/
async function demonstrateActionOrder(
  config,
  tableName,
  key
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Step 1: Create an item with initial values
  const initialItem = {
    ...key,
    counter: 10,
    set_attr: new Set(["A", "B", "C"]),
    to_remove: "This will be removed",
    to_modify: "Original value"
  };

  await docClient.send(new PutCommand({
    TableName: tableName,
    Item: initialItem
  }));

  // Step 2: Get the item to verify initial state
  const responseBefore = await docClient.send(new GetCommand({
    TableName: tableName,
    Key: key
  }));

  const itemBefore = responseBefore.Item || {};

  // Step 3: Update the item with multiple action types
  // The actions will be processed in this order: REMOVE, SET, ADD, DELETE
  const updateResponse = await docClient.send(new UpdateCommand({
    TableName: tableName,
    Key: key,
    UpdateExpression: "REMOVE to_remove SET to_modify = :new_value ADD
counter :increment DELETE set_attr :elements",
    ExpressionAttributeValues: {
      ":new_value": "Updated value",
```

```

        ":increment": 5,
        ":elements": new Set(["B"])
    },
    ReturnValues: "UPDATED_NEW"
  }));

// Step 4: Get the item to verify final state
const responseAfter = await docClient.send(new GetCommand({
  TableName: tableName,
  Key: key
}));

const itemAfter = responseAfter.Item || {};

// Return the results
return {
  initialState: itemBefore,
  updateResponse: updateResponse,
  finalState: itemAfter
};
}

/**
 * Update multiple attributes with a single SET action.
 *
 * This function demonstrates how to update multiple attributes in a single SET
 * action,
 * which is more efficient than using multiple separate update operations.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The primary key of the item to update
 * @param {Object} attributes - The attributes to update and their new values
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateWithMultipleSetActions(
  config,
  tableName,
  key,
  attributes
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

```

```

// Build the update expression and expression attribute values
let updateExpression = "SET ";
const expressionAttributeValues = {};

// Add each attribute to the update expression
Object.entries(attributes).forEach(([attrName, attrValue], index) => {
  const valuePlaceholder = `:val${index}`;

  if (index > 0) {
    updateExpression += ", ";
  }
  updateExpression += `${attrName} = ${valuePlaceholder}`;

  expressionAttributeValues[valuePlaceholder] = attrValue;
});

// Execute the update
const response = await docClient.send(new UpdateCommand({
  TableName: tableName,
  Key: key,
  UpdateExpression: updateExpression,
  ExpressionAttributeValues: expressionAttributeValues,
  ReturnValues: "UPDATED_NEW"
})));

return response;
}

/**
 * Update an attribute with a value from another attribute or a default value.
 *
 * This function demonstrates how to use if_not_exists to conditionally copy a value
 * from one attribute to another, or use a default value if the source doesn't
 * exist.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The primary key of the item to update
 * @param {string} sourceAttribute - The attribute to copy the value from
 * @param {string} targetAttribute - The attribute to update
 * @param {any} defaultValue - The default value to use if the source attribute
 * doesn't exist
 * @returns {Promise<Object>} - The response from DynamoDB

```

```

*/
async function updateWithConditionalValueCopying(
  config,
  tableName,
  key,
  sourceAttribute,
  targetAttribute,
  defaultValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Use if_not_exists to conditionally copy the value
  const response = await docClient.send(new UpdateCommand({
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${targetAttribute} =
if_not_exists(${sourceAttribute}, :default)`,
    ExpressionAttributeValues: {
      ":default": defaultValue
    },
    ReturnValues: "UPDATED_NEW"
  }));

  return response;
}

/**
 * Demonstrate complex update expressions with multiple operations on the same
 * attribute.
 *
 * This function shows how DynamoDB processes multiple operations on the same
 * attribute
 * in a single update expression.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The primary key of the item to create and update
 * @returns {Promise<Object>} - A dictionary containing the results of the
 * demonstration
 */
async function demonstrateMultipleOperationsOnSameAttribute(
  config,

```

```
    tableName,
    key
  ) {
    // Initialize the DynamoDB client
    const client = new DynamoDBClient(config);
    const docClient = DynamoDBDocumentClient.from(client);

    // Step 1: Create an item with initial values
    const initialItem = {
      ...key,
      counter: 10,
      list_attr: [1, 2, 3],
      map_attr: {
        nested1: "value1",
        nested2: "value2"
      }
    };

    await docClient.send(new PutCommand({
      TableName: tableName,
      Item: initialItem
    }));

    // Step 2: Get the item to verify initial state
    const responseBefore = await docClient.send(new GetCommand({
      TableName: tableName,
      Key: key
    }));

    const itemBefore = responseBefore.Item || {};

    // Step 3: Update the item with multiple operations on the same attributes
    const updateResponse = await docClient.send(new UpdateCommand({
      TableName: tableName,
      Key: key,
      UpdateExpression: `
        SET counter = counter + :inc1,
            counter = counter + :inc2,
            map_attr.nested1 = :new_val1,
            map_attr.nested3 = :new_val3,
            list_attr[0] = list_attr[1],
            list_attr[1] = list_attr[2]
      `,
      ExpressionAttributeValues: {
```

```

        ":inc1": 5,
        ":inc2": 3,
        ":new_val1": "updated_value1",
        ":new_val3": "new_value3"
    },
    ReturnValues: "UPDATED_NEW"
  }));

// Step 4: Get the item to verify final state
const responseAfter = await docClient.send(new GetCommand({
  TableName: tableName,
  Key: key
}));

const itemAfter = responseAfter.Item || {};

// Return the results
return {
  initialState: itemBefore,
  updateResponse: updateResponse,
  finalState: itemAfter
};
}

```

Beispiel für die Verwendung von Update Expression Order with AWS SDK für JavaScript.

```

/**
 * Example of how to use update expression order of operations in DynamoDB.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "OrderProcessing";

  console.log("Demonstrating update expression order of operations in DynamoDB");

  try {
    // Example 1: Demonstrating value copying in update expressions
    console.log("\nExample 1: Demonstrating value copying in update expressions");
    const results1 = await demonstrateValueCopying(
      config,
      tableName,

```

```
    { OrderId: "order123" }
  );

  console.log("Initial state:", JSON.stringify(results1.initialState, null, 2));
  console.log("Update response:", JSON.stringify(results1.updateResponse, null,
2));
  console.log("Final state:", JSON.stringify(results1.finalState, null, 2));

  console.log("\nExplanation:");
  console.log("1. The initial state had a=1, b=2, c=3");
  console.log("2. The update expression 'REMOVE a SET b = a, c = b' did the
following:");
  console.log("    - Copied the value of 'a' (which was 1) to be used for 'b'");
  console.log("    - Copied the value of 'b' (which was 2) to be used for 'c'");
  console.log("    - Removed the attribute 'a'");
  console.log("3. The final state has b=1, c=2, and 'a' is removed");
  console.log("4. This demonstrates that DynamoDB uses the values of attributes as
they were BEFORE any modifications");

  // Example 2: Demonstrating the order of different action types
  console.log("\nExample 2: Demonstrating the order of different action types");
  const results2 = await demonstrateActionOrder(
    config,
    tableName,
    { OrderId: "order456" }
  );

  console.log("Initial state:", JSON.stringify(results2.initialState, null, 2));
  console.log("Update response:", JSON.stringify(results2.updateResponse, null,
2));
  console.log("Final state:", JSON.stringify(results2.finalState, null, 2));

  console.log("\nExplanation:");
  console.log("1. The update expression contained multiple action types: REMOVE,
SET, ADD, DELETE");
  console.log("2. DynamoDB processes these actions in this order: REMOVE, SET,
ADD, DELETE");
  console.log("3. First, 'to_remove' was removed");
  console.log("4. Then, 'to_modify' was set to a new value");
  console.log("5. Next, 'counter' was incremented by 5");
  console.log("6. Finally, 'B' was removed from the set attribute");

  // Example 3: Updating multiple attributes in a single SET action
  console.log("\nExample 3: Updating multiple attributes in a single SET action");
```

```
const response3 = await updateWithMultipleSetActions(
  config,
  tableName,
  { OrderId: "order789" },
  {
    Status: "Shipped",
    ShippingDate: "2025-05-28",
    TrackingNumber: "1Z999AA10123456784"
  }
);

console.log("Multiple attributes updated successfully:",
JSON.stringify(response3.Attributes, null, 2));

// Example 4: Conditional value copying with if_not_exists
console.log("\nExample 4: Conditional value copying with if_not_exists");
const response4 = await updateWithConditionalValueCopying(
  config,
  tableName,
  { OrderId: "order101" },
  "PreferredShippingMethod",
  "ShippingMethod",
  "Standard"
);

console.log("Conditional value copying result:",
JSON.stringify(response4.Attributes, null, 2));

// Example 5: Multiple operations on the same attribute
console.log("\nExample 5: Multiple operations on the same attribute");
const results5 = await demonstrateMultipleOperationsOnSameAttribute(
  config,
  tableName,
  { OrderId: "order202" }
);

console.log("Initial state:", JSON.stringify(results5.initialState, null, 2));
console.log("Update response:", JSON.stringify(results5.updateResponse, null,
2));
console.log("Final state:", JSON.stringify(results5.finalState, null, 2));

console.log("\nExplanation:");
console.log("1. The counter was incremented twice (first by 5, then by 3) for a
total of +8");
```



```
    console.log("2. The map attribute had one value updated and a new nested attribute added");
    console.log("3. The list attribute had values shifted (value at index 1 moved to index 0, value at index 2 moved to index 1)");
    console.log("4. All operations within the SET action are processed from left to right");

    // Key points about update expression order of operations
    console.log("\nKey Points About Update Expression Order of Operations:");
    console.log("1. Variables in expressions hold copies of attribute values as they existed BEFORE any modifications");
    console.log("2. Multiple actions in an update expression are processed in this order: REMOVE, SET, ADD, DELETE");
    console.log("3. Within each action type, operations are processed from left to right");
    console.log("4. You can reference the same attribute multiple times in an expression");
    console.log("5. You can use if_not_exists() to conditionally set values based on attribute existence");
    console.log("6. Using a single update expression with multiple actions is more efficient than multiple separate updates");
    console.log("7. The update expression is atomic - either all actions succeed or none do");

    } catch (error) {
        console.error("Error:", error);
    }
}
```

- Einzelheiten zur API finden Sie [UpdateItem](#) unter AWS SDK für JavaScript API-Referenz.

Aktualisieren der Einstellung für den Warmdurchsatz einer Tabelle

Im folgenden Codebeispiel wird gezeigt, wie die Warmdurchsatzeinstellung einer Tabelle aktualisiert wird.

SDK für JavaScript (v3)

Aktualisieren Sie mit AWS SDK für JavaScript die Einstellung für den Warmdurchsatz einer DynamoDB-Tabelle.

```
import { DynamoDBClient, UpdateTableCommand } from "@aws-sdk/client-dynamodb";
```

```
export async function updateDynamoDBTableWarmThroughput(
  tableName,
  tableReadUnits,
  tableWriteUnits,
  gsiName,
  gsiReadUnits,
  gsiWriteUnits,
  region = "us-east-1"
) {
  try {
    const ddbClient = new DynamoDBClient({ region: region });

    // Construct the update table request
    const updateTableRequest = {
      TableName: tableName,
      GlobalSecondaryIndexUpdates: [
        {
          Update: {
            IndexName: gsiName,
            WarmThroughput: {
              ReadUnitsPerSecond: gsiReadUnits,
              WriteUnitsPerSecond: gsiWriteUnits,
            },
          },
        },
      ],
      WarmThroughput: {
        ReadUnitsPerSecond: tableReadUnits,
        WriteUnitsPerSecond: tableWriteUnits,
      },
    };

    const command = new UpdateTableCommand(updateTableRequest);
    const response = await ddbClient.send(command);
    console.log(`Table updated successfully! Response:
    ${JSON.stringify(response)}`);
    return response;
  } catch (error) {
    console.error(`Error updating table: ${error}`);
    throw error;
  }
}
```

```
// Example usage (commented out for testing)
/*
updateDynamoDBTableWarmThroughput(
  'example-table',
  5, 5,
  'example-index',
  2, 2
);
*/
```

- Einzelheiten zur API finden Sie [UpdateTable](#) in der AWS SDK für JavaScript API-Referenz.

Aktualisieren der TTL eines Elements

Das folgende Codebeispiel zeigt, wie Sie die TTL eines Elements aktualisieren.

SDK für JavaScript (v3)

```
import { DynamoDBClient, UpdateItemCommand } from "@aws-sdk/client-dynamodb";
import { marshall, unmarshall } from "@aws-sdk/util-dynamodb";

export const updateItem = async (tableName, partitionKey, sortKey, region = 'us-east-1') => {
  const client = new DynamoDBClient({
    region: region,
    endpoint: `https://dynamodb.${region}.amazonaws.com`
  });

  const currentTime = Math.floor(Date.now() / 1000);
  const expireAt = Math.floor((Date.now() + 90 * 24 * 60 * 60 * 1000) / 1000);

  const params = {
    TableName: tableName,
    Key: marshall({
      partitionKey: partitionKey,
      sortKey: sortKey
    }),
    UpdateExpression: "SET updatedAt = :c, expireAt = :e",
    ExpressionAttributeValues: marshall({
      ":c": currentTime,
      ":e": expireAt
    })
  };
}
```

```

    }},
  };

  try {
    const data = await client.send(new UpdateItemCommand(params));
    const responseData = unmarshall(data.Attributes);
    console.log("Item updated successfully: %s", responseData);
    return responseData;
  } catch (err) {
    console.error("Error updating item:", err);
    throw err;
  }
}

// Example usage (commented out for testing)
// updateItem('your-table-name', 'your-partition-key-value', 'your-sort-key-value');

```

- Einzelheiten zur API finden Sie [UpdateItem](#) in der AWS SDK für JavaScript API-Referenz.

Aktualisieren von Daten mit PartiQL UPDATE

Im folgenden Codebeispiel wird gezeigt, wie Sie Daten mit PartiQL-UPDATE-Anweisungen aktualisieren.

SDK für JavaScript (v3)

Aktualisieren Sie Elemente in einer DynamoDB-Tabelle mithilfe von PartiQL UPDATE-Anweisungen mit AWS SDK für JavaScript

```

/**
 * This example demonstrates how to update items in a DynamoDB table using PartiQL.
 * It shows different ways to update documents with various index types.
 */
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  DynamoDBDocumentClient,
  ExecuteStatementCommand,
  BatchExecuteStatementCommand,
} from "@aws-sdk/lib-dynamodb";

```

```

/**
 * Update a single attribute of an item using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @param attributeName - The name of the attribute to update
 * @param attributeValue - The new value for the attribute
 * @returns The response from the ExecuteStatementCommand
 */
export const updateSingleAttribute = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number,
  attributeName: string,
  attributeValue: any
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `UPDATE "${tableName}" SET ${attributeName} = ? WHERE
${partitionKeyName} = ?`,
    Parameters: [attributeValue, partitionKeyValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item updated successfully");
    return data;
  } catch (err) {
    console.error("Error updating item:", err);
    throw err;
  }
};

/**
 * Update multiple attributes of an item using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @param attributeUpdates - Object containing attribute names and their new values
 * @returns The response from the ExecuteStatementCommand

```

```

*/
export const updateMultipleAttributes = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number,
  attributeUpdates: Record<string, any>
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  // Create SET clause for each attribute
  const setClause = Object.keys(attributeUpdates)
    .map((attr, index) => `${attr} = ?`)
    .join(", ");

  // Create parameters array with attribute values followed by the partition key
  // value
  const parameters = [...Object.values(attributeUpdates), partitionKeyValue];

  const params = {
    Statement: `UPDATE "${tableName}" SET ${setClause} WHERE ${partitionKeyName} = ?`,
    Parameters: parameters,
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item updated successfully");
    return data;
  } catch (err) {
    console.error("Error updating item:", err);
    throw err;
  }
};

/**
 * Update an item identified by a composite key (partition key + sort key) using
 * PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param partitionKeyName - The name of the partition key attribute
 * @param partitionKeyValue - The value of the partition key
 * @param sortKeyName - The name of the sort key attribute
 * @param sortKeyValue - The value of the sort key

```

```

* @param attributeName - The name of the attribute to update
* @param attributeValue - The new value for the attribute
* @returns The response from the ExecuteStatementCommand
*/
export const updateItemWithCompositeKey = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number,
  sortKeyName: string,
  sortKeyValue: string | number,
  attributeName: string,
  attributeValue: any
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `UPDATE "${tableName}" SET ${attributeName} = ? WHERE
${partitionKeyName} = ? AND ${sortKeyName} = ?`,
    Parameters: [attributeValue, partitionKeyValue, sortKeyValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item updated successfully");
    return data;
  } catch (err) {
    console.error("Error updating item:", err);
    throw err;
  }
};

/**
* Update an item with a condition to ensure the update only happens if a condition
is met.
*
* @param tableName - The name of the DynamoDB table
* @param partitionKeyName - The name of the partition key attribute
* @param partitionKeyValue - The value of the partition key
* @param attributeName - The name of the attribute to update
* @param attributeValue - The new value for the attribute
* @param conditionAttribute - The attribute to check in the condition
* @param conditionValue - The value to compare against in the condition
* @returns The response from the ExecuteStatementCommand

```

```

*/
export const updateItemWithCondition = async (
  tableName: string,
  partitionKeyName: string,
  partitionKeyValue: string | number,
  attributeName: string,
  attributeValue: any,
  conditionAttribute: string,
  conditionValue: any
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  const params = {
    Statement: `UPDATE "${tableName}" SET ${attributeName} = ? WHERE
${partitionKeyName} = ? AND ${conditionAttribute} = ?`,
    Parameters: [attributeValue, partitionKeyValue, conditionValue],
  };

  try {
    const data = await docClient.send(new ExecuteStatementCommand(params));
    console.log("Item updated with condition successfully");
    return data;
  } catch (err) {
    console.error("Error updating item with condition:", err);
    throw err;
  }
};

/**
 * Batch update multiple items using PartiQL.
 *
 * @param tableName - The name of the DynamoDB table
 * @param updates - Array of objects containing key and update information
 * @returns The response from the BatchExecuteStatementCommand
 */
export const batchUpdateItems = async (
  tableName: string,
  updates: Array<{
    partitionKeyName: string;
    partitionKeyValue: string | number;
    attributeName: string;
    attributeValue: any;
  }>
)

```



```
) => {
  const client = new DynamoDBClient({});
  const docClient = DynamoDBDocumentClient.from(client);

  // Create statements for each update
  const statements = updates.map((update) => {
    return {
      Statement: `UPDATE "${tableName}" SET ${update.attributeName} = ? WHERE
${update.partitionKeyName} = ?`,
      Parameters: [update.attributeValue, update.partitionKeyValue],
    };
  });

  const params = {
    Statements: statements,
  };

  try {
    const data = await docClient.send(new BatchExecuteStatementCommand(params));
    console.log("Items batch updated successfully");
    return data;
  } catch (err) {
    console.error("Error batch updating items:", err);
    throw err;
  }
};

/**
 * Example usage showing how to update items with different index types
 */
export const updateExamples = async () => {
  // Update a single attribute using a simple primary key
  await updateSingleAttribute("UsersTable", "userId", "user123", "email",
    "newemail@example.com");

  // Update multiple attributes at once
  await updateMultipleAttributes("UsersTable", "userId", "user123", {
    email: "newemail@example.com",
    name: "John Smith",
    lastLogin: new Date().toISOString(),
  });

  // Update an item with a composite key (partition key + sort key)
  await updateItemWithCompositeKey(
```

```
    "OrdersTable",
    "orderId",
    "order456",
    "productId",
    "prod789",
    "quantity",
    5
  );

// Update with a condition
await updateItemWithCondition(
  "UsersTable",
  "userId",
  "user123",
  "userStatus",
  "active",
  "userType",
  "premium"
);

// Batch update multiple items
await batchUpdateItems("UsersTable", [
  {
    partitionKeyName: "userId",
    partitionKeyValue: "user123",
    attributeName: "lastLogin",
    attributeValue: new Date().toISOString(),
  },
  {
    partitionKeyName: "userId",
    partitionKeyValue: "user456",
    attributeName: "lastLogin",
    attributeValue: new Date().toISOString(),
  },
]);
};
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für JavaScript - API-Referenz.
 - [BatchExecuteStatement](#)
 - [ExecuteStatement](#)

Verwenden von API Gateway zum Aufrufen einer Lambda-Funktion

Das folgende Codebeispiel zeigt, wie eine von Amazon API Gateway aufgerufene AWS Lambda Funktion erstellt wird.

SDK für JavaScript (v3)

Zeigt, wie eine AWS Lambda Funktion mithilfe der JavaScript Lambda-Laufzeit-API erstellt wird. In diesem Beispiel werden verschiedene AWS Dienste aufgerufen, um einen bestimmten Anwendungsfall auszuführen. Dieses Beispiel zeigt, wie man eine Lambda-Funktion erstellt, die von Amazon API Gateway aufgerufen wird und eine Amazon-DynamoDB-Tabelle nach Arbeitsjubiläen durchsucht und Amazon Simple Notification Service (Amazon SNS) verwendet, um eine Textnachricht an Ihre Mitarbeiter zu senden, die ihnen zu ihrem einjährigen Jubiläum gratuliert.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

Dieses Beispiel ist auch verfügbar im [AWS SDK für JavaScript Entwicklerhandbuch für v3](#).

In diesem Beispiel verwendete Dienste

- API Gateway
- DynamoDB
- Lambda
- Amazon SNS

Verwenden von Operationen mit unteilbaren Zählern

Im folgenden Codebeispiel wird gezeigt, wie Sie Operationen mit unteilbaren Zählern in DynamoDB verwenden.

- Inkrementieren von unteilbaren Zählern mithilfe von ADD- und SET-Operationen
- Sicheres Erhöhen von Zählern, die möglicherweise nicht existieren
- Implementieren einer optimistischen Sperre für Zähleroperationen

SDK für JavaScript (v3)

Demonstrieren Sie atomare Gegenoperationen mit AWS SDK für JavaScript.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  GetCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Increment a counter using the ADD operation.
 *
 * This function demonstrates using the ADD operation for atomic increments.
 * The ADD operation is atomic and is the recommended way to increment counters.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} counterName - The name of the counter attribute
 * @param {number} incrementValue - The value to increment by
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function incrementCounterWithAdd(
  config,
  tableName,
  key,
  counterName,
  incrementValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using ADD
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `ADD ${counterName} :increment`,
    ExpressionAttributeValues: {
      ":increment": incrementValue
    },
    ReturnValues: "UPDATED_NEW"
  };

  // Perform the update operation
```

```
    const response = await docClient.send(new UpdateCommand(params));

    return response;
}

/**
 * Increment a counter using the SET operation with an expression.
 *
 * This function demonstrates using the SET operation with an expression for
 * increments.
 * While this approach works, it's less idiomatic for simple increments than using
 * ADD.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} counterName - The name of the counter attribute
 * @param {number} incrementValue - The value to increment by
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function incrementCounterWithSet(
    config,
    tableName,
    key,
    counterName,
    incrementValue
) {
    // Initialize the DynamoDB client
    const client = new DynamoDBClient(config);
    const docClient = DynamoDBDocumentClient.from(client);

    // Define the update parameters using SET with an expression
    const params = {
        TableName: tableName,
        Key: key,
        UpdateExpression: `SET ${counterName} = ${counterName} + :increment`,
        ExpressionAttributeValues: {
            ":increment": incrementValue
        },
        ReturnValues: "UPDATED_NEW"
    };

    // Perform the update operation
    const response = await docClient.send(new UpdateCommand(params));
}
```

```

    return response;
}

/**
 * Increment a counter safely, handling the case where the counter might not exist.
 *
 * This function demonstrates using the if_not_exists function with SET to safely
 * increment a counter that might not exist yet.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} counterName - The name of the counter attribute
 * @param {number} incrementValue - The value to increment by
 * @param {number} defaultValue - The default value if the counter doesn't exist
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function incrementCounterSafely(
    config,
    tableName,
    key,
    counterName,
    incrementValue,
    defaultValue = 0
) {
    // Initialize the DynamoDB client
    const client = new DynamoDBClient(config);
    const docClient = DynamoDBDocumentClient.from(client);

    // Define the update parameters using SET with if_not_exists
    const params = {
        TableName: tableName,
        Key: key,
        UpdateExpression: `SET ${counterName} = if_not_exists(${counterName}, :default)
+ :increment`,
        ExpressionAttributeValues: {
            ":increment": incrementValue,
            ":default": defaultValue
        },
        ReturnValues: "UPDATED_NEW"
    };

    // Perform the update operation

```

```

    const response = await docClient.send(new UpdateCommand(params));

    return response;
}

/**
 * Increment a counter with optimistic locking to prevent race conditions.
 *
 * This function demonstrates using a condition expression to implement optimistic
 * locking, which prevents race conditions when multiple processes try to update
 * the same counter.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} counterName - The name of the counter attribute
 * @param {number} incrementValue - The value to increment by
 * @param {number} expectedValue - The expected current value of the counter
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function incrementCounterWithLocking(
    config,
    tableName,
    key,
    counterName,
    incrementValue,
    expectedValue
) {
    // Initialize the DynamoDB client
    const client = new DynamoDBClient(config);
    const docClient = DynamoDBDocumentClient.from(client);

    // Define the update parameters with a condition expression
    const params = {
        TableName: tableName,
        Key: key,
        UpdateExpression: `SET ${counterName} = ${counterName} + :increment`,
        ConditionExpression: `${counterName} = :expected`,
        ExpressionAttributeValues: {
            ":increment": incrementValue,
            ":expected": expectedValue
        },
        ReturnValues: "UPDATED_NEW"
    };
};

```

```

    try {
      // Perform the update operation
      const response = await docClient.send(new UpdateCommand(params));
      return {
        success: true,
        data: response
      };
    } catch (error) {
      // Check if the error is due to the condition check failing
      if (error.name === "ConditionalCheckFailedException") {
        return {
          success: false,
          error: "Optimistic locking failed: the counter value has changed"
        };
      }
      // Re-throw other errors
      throw error;
    }
  }
}

/**
 * Get the current value of a counter.
 *
 * Helper function to retrieve the current value of a counter attribute.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to get
 * @param {string} counterName - The name of the counter attribute
 * @returns {Promise<number|null>} - The current counter value or null if not found
 */
async function getCounterValue(
  config,
  tableName,
  key,
  counterName
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the get parameters
  const params = {

```



```
    TableName: tableName,
    Key: key
  };

  // Perform the get operation
  const response = await docClient.send(new GetCommand(params));

  // Return the counter value if it exists, otherwise null
  return response.Item && counterName in response.Item
    ? response.Item[counterName]
    : null;
}
```

Beispiel für die Verwendung atomarer Zähleroperationen mit AWS SDK für JavaScript.

```
/**
 * Example of how to use the atomic counter operations.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "Products";
  const key = { ProductId: "P12345" };
  const counterName = "ViewCount";
  const incrementValue = 1;

  console.log("Demonstrating different approaches to increment counters in
  DynamoDB");

  try {
    // Example 1: Using ADD operation (recommended for simple increments)
    console.log("\nExample 1: Incrementing counter with ADD operation");
    const response1 = await incrementCounterWithAdd(
      config,
      tableName,
      key,
      counterName,
      incrementValue
    );

    console.log(`Counter incremented to: ${response1.Attributes[counterName]}`);
  }
}
```

```
// Example 2: Using SET operation with an expression
console.log("\nExample 2: Incrementing counter with SET operation");
const response2 = await incrementCounterWithSet(
  config,
  tableName,
  key,
  counterName,
  incrementValue
);

console.log(`Counter incremented to: ${response2.Attributes[counterName]}`);

// Example 3: Safely incrementing a counter that might not exist
console.log("\nExample 3: Safely incrementing counter that might not exist");
const newKey = { ProductId: "P67890" };
const response3 = await incrementCounterSafely(
  config,
  tableName,
  newKey,
  counterName,
  incrementValue,
  0
);

console.log(`Counter initialized and incremented to:
${response3.Attributes[counterName]}`);

// Example 4: Incrementing with optimistic locking
console.log("\nExample 4: Incrementing with optimistic locking");

// First, get the current counter value
const currentValue = await getCounterValue(config, tableName, key, counterName);
console.log(`Current counter value: ${currentValue}`);

// Then, try to increment with optimistic locking
const response4 = await incrementCounterWithLocking(
  config,
  tableName,
  key,
  counterName,
  incrementValue,
  currentValue
);
```

```
    if (response4.success) {
        console.log(`Counter successfully incremented to:
${response4.data.Attributes[counterName]}`);
    } else {
        console.log(response4.error);
    }

    // Explain the differences between ADD and SET
    console.log("\nKey differences between ADD and SET for counter operations:");
    console.log("1. ADD is more concise and idiomatic for simple increments");
    console.log("2. SET with expressions is more flexible for complex operations");
    console.log("3. Both operations are atomic and safe for concurrent updates");
    console.log("4. SET with if_not_exists is required when the attribute might not
exist");
    console.log("5. Optimistic locking can be added to either approach for
additional safety");

    } catch (error) {
        console.error("Error:", error);
    }
}
```

- Einzelheiten zur API finden Sie [UpdateItem](#) in der AWS SDK für JavaScript API-Referenz.

Verwenden von bedingten Operationen

Im folgenden Codebeispiel wird gezeigt, wie bedingte Operationen in DynamoDB verwendet werden.

- Implementieren von bedingten Schreibvorgängen, um das Überschreiben von Daten zu verhindern
- Verwenden von Bedingungsausdrücken, um Geschäftsregeln durchzusetzen
- Ordnungsgemäßer Umgang mit Fehlern bei bedingten Prüfungen

SDK für JavaScript (v3)

Demonstrieren Sie bedingte Operationen mit AWS SDK für JavaScript.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
const {
    DynamoDBDocumentClient,
    UpdateCommand,
```

```

    DeleteCommand,
    GetCommand,
    PutCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Perform a conditional update operation.
 *
 * This function demonstrates how to update an item only if a condition is met.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} conditionAttribute - The attribute to check in the condition
 * @param {any} conditionValue - The value to compare against
 * @param {string} updateAttribute - The attribute to update
 * @param {any} updateValue - The new value to set
 * @returns {Promise<Object>} - Result of the operation
 */
async function conditionalUpdate(
  config,
  tableName,
  key,
  conditionAttribute,
  conditionValue,
  updateAttribute,
  updateValue
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters with a condition expression
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${updateAttribute} = :value`,
    ConditionExpression: `${conditionAttribute} = :condition`,
    ExpressionAttributeValues: {
      ":value": updateValue,
      ":condition": conditionValue
    },
    ReturnValues: "UPDATED_NEW"
  };
};

```

```

    try {
      // Perform the update operation
      const response = await docClient.send(new UpdateCommand(params));

      return {
        success: true,
        message: "Condition was met and update was performed",
        updatedAttributes: response.Attributes
      };
    } catch (error) {
      // Check if the error is due to the condition check failing
      if (error.name === "ConditionalCheckFailedException") {
        return {
          success: false,
          message: "Condition was not met, update was not performed",
          error: "ConditionalCheckFailedException"
        };
      }

      // Re-throw other errors
      throw error;
    }
  }

  /**
   * Perform a conditional delete operation.
   *
   * This function demonstrates how to delete an item only if a condition is met.
   *
   * @param {Object} config - AWS configuration object
   * @param {string} tableName - The name of the DynamoDB table
   * @param {Object} key - The key of the item to delete
   * @param {string} conditionAttribute - The attribute to check in the condition
   * @param {any} conditionValue - The value to compare against
   * @returns {Promise<Object>} - Result of the operation
   */
  async function conditionalDelete(
    config,
    tableName,
    key,
    conditionAttribute,
    conditionValue
  ) {

```

```
// Initialize the DynamoDB client
const client = new DynamoDBClient(config);
const docClient = DynamoDBDocumentClient.from(client);

// Define the delete parameters with a condition expression
const params = {
  TableName: tableName,
  Key: key,
  ConditionExpression: `${conditionAttribute} = :condition`,
  ExpressionAttributeValues: {
    ":condition": conditionValue
  },
  ReturnValues: "ALL_OLD"
};

try {
  // Perform the delete operation
  const response = await docClient.send(new DeleteCommand(params));

  return {
    success: true,
    message: "Condition was met and item was deleted",
    deletedItem: response.Attributes
  };
} catch (error) {
  // Check if the error is due to the condition check failing
  if (error.name === "ConditionalCheckFailedException") {
    return {
      success: false,
      message: "Condition was not met, item was not deleted",
      error: "ConditionalCheckFailedException"
    };
  }

  // Re-throw other errors
  throw error;
}

/**
 * Implement optimistic locking with a version number.
 *
 * This function demonstrates how to use a version number for optimistic locking
 * to prevent race conditions when multiple processes update the same item.
 */
```

```
*
* @param {Object} config - AWS configuration object
* @param {string} tableName - The name of the DynamoDB table
* @param {Object} key - The key of the item to update
* @param {Object} updates - The attributes to update
* @param {number} expectedVersion - The expected current version number
* @returns {Promise<Object>} - Result of the operation
*/
async function updateWithOptimisticLocking(
  config,
  tableName,
  key,
  updates,
  expectedVersion
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Build the update expression
  const updateExpressions = [];
  const expressionAttributeValues = {
    ":expectedVersion": expectedVersion,
    ":newVersion": expectedVersion + 1
  };

  // Add each update to the expression
  Object.entries(updates).forEach(([attribute, value], index) => {
    updateExpressions.push(`${attribute} = :val${index}`);
    expressionAttributeValues[`val${index}`] = value;
  });

  // Add the version update
  updateExpressions.push("version = :newVersion");

  // Define the update parameters with a condition expression
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${updateExpressions.join(", ")}`,
    ConditionExpression: "version = :expectedVersion",
    ExpressionAttributeValues: expressionAttributeValues,
    ReturnValues: "UPDATED_NEW"
  };
};
```

```

try {
  // Perform the update operation
  const response = await docClient.send(new UpdateCommand(params));

  return {
    success: true,
    message: "Update succeeded with optimistic locking",
    newVersion: expectedVersion + 1,
    updatedAttributes: response.Attributes
  };
} catch (error) {
  // Check if the error is due to the condition check failing
  if (error.name === "ConditionalCheckFailedException") {
    return {
      success: false,
      message: "Optimistic locking failed: the item was modified by another
process",
      error: "ConditionalCheckFailedException"
    };
  }

  // Re-throw other errors
  throw error;
}
}

/**
 * Implement a conditional write that creates an item only if it doesn't exist.
 *
 * This function demonstrates how to use attribute_not_exists to create an item
 * only if it doesn't already exist (similar to an "INSERT IF NOT EXISTS"
 * operation).
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} item - The item to create
 * @returns {Promise<Object>} - Result of the operation
 */
async function createIfNotExists(
  config,
  tableName,
  item
) {

```



```
// Initialize the DynamoDB client
const client = new DynamoDBClient(config);
const docClient = DynamoDBDocumentClient.from(client);

// Extract the primary key attributes
const keyAttributes = Object.keys(item).filter(attr =>
  attr === "id" || attr === "ID" || attr === "Id" ||
  attr.endsWith("Id") || attr.endsWith("ID") ||
  attr.endsWith("Key")
);

if (keyAttributes.length === 0) {
  throw new Error("Could not determine primary key attributes");
}

// Create a condition expression that checks if the item doesn't exist
const conditionExpression = `attribute_not_exists(${keyAttributes[0]})`;

// Define the put parameters with a condition expression
const params = {
  TableName: tableName,
  Item: item,
  ConditionExpression: conditionExpression
};

try {
  // Perform the put operation
  await docClient.send(new PutCommand(params));

  return {
    success: true,
    message: "Item was created because it didn't exist",
    item
  };
} catch (error) {
  // Check if the error is due to the condition check failing
  if (error.name === "ConditionalCheckFailedException") {
    return {
      success: false,
      message: "Item already exists, creation was skipped",
      error: "ConditionalCheckFailedException"
    };
  }
}
```

```
        // Re-throw other errors
        throw error;
    }
}

/**
 * Get the current value of an item.
 *
 * Helper function to retrieve the current value of an item.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to get
 * @returns {Promise<Object|null>} - The item or null if not found
 */
async function getItem(
    config,
    tableName,
    key
) {
    // Initialize the DynamoDB client
    const client = new DynamoDBClient(config);
    const docClient = DynamoDBDocumentClient.from(client);

    // Define the get parameters
    const params = {
        TableName: tableName,
        Key: key
    };

    // Perform the get operation
    const response = await docClient.send(new GetCommand(params));

    // Return the item if it exists, otherwise null
    return response.Item || null;
}
```

Beispiel für die Verwendung von bedingten Operationen mit AWS SDK für JavaScript.

```
/**
 * Example of how to use conditional operations.
 */
```

```
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "Products";
  const key = { ProductId: "P12345" };

  console.log("Demonstrating conditional operations in DynamoDB");

  try {
    // Example 1: Conditional update based on attribute value
    console.log("\nExample 1: Conditional update based on attribute value");
    const updateResult = await conditionalUpdate(
      config,
      tableName,
      key,
      "Category",
      "Electronics",
      "Price",
      299.99
    );

    console.log(`Result: ${updateResult.message}`);
    if (updateResult.success) {
      console.log("Updated attributes:", updateResult.updatedAttributes);
    }

    // Example 2: Conditional delete based on attribute value
    console.log("\nExample 2: Conditional delete based on attribute value");
    const deleteResult = await conditionalDelete(
      config,
      tableName,
      key,
      "InStock",
      false
    );

    console.log(`Result: ${deleteResult.message}`);
    if (deleteResult.success) {
      console.log("Deleted item:", deleteResult.deletedItem);
    }

    // Example 3: Optimistic locking with version number
    console.log("\nExample 3: Optimistic locking with version number");
  }
}
```

```
// First, get the current item to check its version
const currentItem = await getItem(config, tableName, { ProductId: "P67890" });
const currentVersion = currentItem ? (currentItem.version || 0) : 0;

console.log(`Current version: ${currentVersion}`);

// Then, update with optimistic locking
const lockingResult = await updateWithOptimisticLocking(
  config,
  tableName,
  { ProductId: "P67890" },
  {
    Name: "Updated Product Name",
    Description: "This is an updated description"
  },
  currentVersion
);

console.log(`Result: ${lockingResult.message}`);
if (lockingResult.success) {
  console.log(`New version: ${lockingResult.newVersion}`);
  console.log("Updated attributes:", lockingResult.updatedAttributes);
}

// Example 4: Create item only if it doesn't exist
console.log("\nExample 4: Create item only if it doesn't exist");
const createResult = await createIfNotExists(
  config,
  tableName,
  {
    ProductId: "P99999",
    Name: "New Product",
    Category: "Accessories",
    Price: 19.99,
    InStock: true
  }
);

console.log(`Result: ${createResult.message}`);
if (createResult.success) {
  console.log("Created item:", createResult.item);
}

// Explain conditional operations
```

```
console.log("\nKey points about conditional operations:");
console.log("1. Conditional operations only succeed if the condition is met");
console.log("2. ConditionalCheckFailedException indicates the condition wasn't
met");
console.log("3. Optimistic locking prevents race conditions in concurrent
updates");
console.log("4. attribute_exists and attribute_not_exists are useful for
checking if attributes are present");
console.log("5. Conditional operations are atomic - they either succeed
completely or fail completely");
console.log("6. You can use any valid comparison operators and functions in
condition expressions");
console.log("7. Conditional operations don't consume write capacity if the
condition fails");

} catch (error) {
  console.error("Error:", error);
}
}
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für JavaScript - API-Referenz.
 - [DeleteItem](#)
 - [PutItem](#)
 - [UpdateItem](#)

Verwenden von Ausdrucksattributnamen

Im folgenden Codebeispiel wird gezeigt, wie Sie Ausdrucksattributnamen in DynamoDB verwenden.

- Arbeiten mit reservierten Wörtern in DynamoDB-Ausdrücken
- Verwenden von Platzhaltern für Ausdrucksattributnamen
- Behandeln von Sonderzeichen in Attributnamen

SDK für JavaScript (v3)

Demonstrieren Sie die Namen der Ausdrucksattribute mit AWS SDK für JavaScript.

```
const { DynamoDBClient } = require("@aws-sdk/client-dynamodb");
```

```
const {
  DynamoDBDocumentClient,
  UpdateCommand,
  GetCommand,
  QueryCommand,
  ScanCommand
} = require("@aws-sdk/lib-dynamodb");

/**
 * Update an attribute that is a reserved word in DynamoDB.
 *
 * This function demonstrates how to use expression attribute names to update
 * attributes that are reserved words in DynamoDB.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} reservedWordAttribute - The reserved word attribute to update
 * @param {any} value - The value to set
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateReservedWordAttribute(
  config,
  tableName,
  key,
  reservedWordAttribute,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using expression attribute names
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: "SET #attr = :value",
    ExpressionAttributeNames: {
      "#attr": reservedWordAttribute
    },
    ExpressionAttributeValues: {
      ":value": value
    },
    ReturnValues: "UPDATED_NEW"
  };
}
```

```
};

// Perform the update operation
const response = await docClient.send(new UpdateCommand(params));

return response;
}

/**
 * Update an attribute that contains special characters.
 *
 * This function demonstrates how to use expression attribute names to update
 * attributes that contain special characters.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string} specialCharAttribute - The attribute with special characters to
 * update
 * @param {any} value - The value to set
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateSpecialCharacterAttribute(
  config,
  tableName,
  key,
  specialCharAttribute,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Define the update parameters using expression attribute names
  const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: "SET #attr = :value",
    ExpressionAttributeNames: {
      "#attr": specialCharAttribute
    },
    ExpressionAttributeValues: {
      ":value": value
    },
  },
```

```
        ReturnValues: "UPDATED_NEW"
    };

    // Perform the update operation
    const response = await docClient.send(new UpdateCommand(params));

    return response;
}

/**
 * Query items using an attribute that is a reserved word.
 *
 * This function demonstrates how to use expression attribute names in a query
 * when the attribute is a reserved word.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {string} partitionKeyName - The name of the partition key attribute
 * @param {any} partitionKeyValue - The value of the partition key
 * @param {string} reservedWordAttribute - The reserved word attribute to filter on
 * @param {any} value - The value to compare against
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function queryWithReservedWordAttribute(
    config,
    tableName,
    partitionKeyName,
    partitionKeyValue,
    reservedWordAttribute,
    value
) {
    // Initialize the DynamoDB client
    const client = new DynamoDBClient(config);
    const docClient = DynamoDBDocumentClient.from(client);

    // Define the query parameters using expression attribute names
    const params = {
        TableName: tableName,
        KeyConditionExpression: "#pkName = :pkValue",
        FilterExpression: "#attr = :value",
        ExpressionAttributeNames: {
            "#pkName": partitionKeyName,
            "#attr": reservedWordAttribute
        },
    },
```



```

    ExpressionAttributeValues: {
      ":pkValue": partitionKeyValue,
      ":value": value
    }
  };

  // Perform the query operation
  const response = await docClient.send(new QueryCommand(params));

  return response;
}

/**
 * Update a nested attribute with a path that contains reserved words.
 *
 * This function demonstrates how to use expression attribute names to update
 * nested attributes where the path contains reserved words.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to update
 * @param {string[]} attributePath - The path to the nested attribute as an array
 * @param {any} value - The value to set
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function updateNestedReservedWordAttribute(
  config,
  tableName,
  key,
  attributePath,
  value
) {
  // Initialize the DynamoDB client
  const client = new DynamoDBClient(config);
  const docClient = DynamoDBDocumentClient.from(client);

  // Create expression attribute names for each part of the path
  const expressionAttributeNames = {};
  for (let i = 0; i < attributePath.length; i++) {
    expressionAttributeNames[`#attr${i}`] = attributePath[i];
  }

  // Build the attribute path using the expression attribute names
  const attributePathExpression = attributePath

```

```

        .map( (_, i) => `#attr${i}` )
        .join(".");

// Define the update parameters
const params = {
    TableName: tableName,
    Key: key,
    UpdateExpression: `SET ${attributePathExpression} = :value`,
    ExpressionAttributeNames: expressionAttributeNames,
    ExpressionAttributeValues: {
        ":value": value
    },
    ReturnValues: "UPDATED_NEW"
};

// Perform the update operation
const response = await docClient.send(new UpdateCommand(params));

return response;
}

/**
 * Scan a table with multiple attribute name placeholders.
 *
 * This function demonstrates how to use multiple expression attribute names
 * in a complex filter expression.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} filters - Object mapping attribute names to filter values
 * @returns {Promise<Object>} - The response from DynamoDB
 */
async function scanWithMultipleAttributeNames(
    config,
    tableName,
    filters
) {
    // Initialize the DynamoDB client
    const client = new DynamoDBClient(config);
    const docClient = DynamoDBDocumentClient.from(client);

    // Create expression attribute names and values
    const expressionAttributeNames = {};
    const expressionAttributeValues = {};

```

```

const filterConditions = [];

// Build the filter expression
Object.entries(filters).forEach(([attrName, value], index) => {
  const nameKey = `#attr${index}`;
  const valueKey = `:val${index}`;

  expressionAttributeNames[nameKey] = attrName;
  expressionAttributeValues[valueKey] = value;
  filterConditions.push(`${nameKey} = ${valueKey}`);
});

// Join the filter conditions with AND
const filterExpression = filterConditions.join(" AND ");

// Define the scan parameters
const params = {
  TableName: tableName,
  FilterExpression: filterExpression,
  ExpressionAttributeNames: expressionAttributeNames,
  ExpressionAttributeValues: expressionAttributeValues
};

// Perform the scan operation
const response = await docClient.send(new ScanCommand(params));

return response;
}

/**
 * Get the current value of an item.
 *
 * Helper function to retrieve the current value of an item.
 *
 * @param {Object} config - AWS configuration object
 * @param {string} tableName - The name of the DynamoDB table
 * @param {Object} key - The key of the item to get
 * @returns {Promise<Object|null>} - The item or null if not found
 */
async function getItem(
  config,
  tableName,
  key
) {

```

```
// Initialize the DynamoDB client
const client = new DynamoDBClient(config);
const docClient = DynamoDBDocumentClient.from(client);

// Define the get parameters
const params = {
  TableName: tableName,
  Key: key
};

// Perform the get operation
const response = await docClient.send(new GetCommand(params));

// Return the item if it exists, otherwise null
return response.Item || null;
}
```

Beispiel für die Verwendung von Ausdrucksattributnamen mit AWS SDK für JavaScript.

```
/**
 * Example of how to use expression attribute names.
 */
async function exampleUsage() {
  // Example parameters
  const config = { region: "us-west-2" };
  const tableName = "Products";
  const key = { ProductId: "P12345" };

  console.log("Demonstrating expression attribute names in DynamoDB");

  try {
    // Example 1: Update an attribute that is a reserved word
    console.log("\nExample 1: Updating an attribute that is a reserved word");
    const response1 = await updateReservedWordAttribute(
      config,
      tableName,
      key,
      "Size", // "SIZE" is a reserved word in DynamoDB
      "Large"
    );

    console.log("Updated attribute:", response1.Attributes);
  }
}
```

```
// Example 2: Update an attribute with special characters
console.log("\nExample 2: Updating an attribute with special characters");
const response2 = await updateSpecialCharacterAttribute(
  config,
  tableName,
  key,
  "Product-Type", // Contains a hyphen, which is a special character
  "Electronics"
);

console.log("Updated attribute:", response2.Attributes);

// Example 3: Query with a reserved word attribute
console.log("\nExample 3: Querying with a reserved word attribute");
const response3 = await queryWithReservedWordAttribute(
  config,
  tableName,
  "Category",
  "Electronics",
  "Count", // "COUNT" is a reserved word in DynamoDB
  10
);

console.log(`Found ${response3.Items.length} items`);

// Example 4: Update a nested attribute with reserved words in the path
console.log("\nExample 4: Updating a nested attribute with reserved words in the path");
const response4 = await updateNestedReservedWordAttribute(
  config,
  tableName,
  key,
  ["Dimensions", "Size", "Height"], // "SIZE" is a reserved word
  30
);

console.log("Updated nested attribute:", response4.Attributes);

// Example 5: Scan with multiple attribute name placeholders
console.log("\nExample 5: Scanning with multiple attribute name placeholders");
const response5 = await scanWithMultipleAttributeNames(
  config,
  tableName,
```

```
{
  "Size": "Large",
  "Count": 10,
  "Product-Type": "Electronics"
}
);

console.log(`Found ${response5.Items.length} items`);

// Get the final state of the item
console.log("\nFinal state of the item:");
const item = await getItem(config, tableName, key);
console.log(JSON.stringify(item, null, 2));

// Show some common reserved words
console.log("\nSome common DynamoDB reserved words:");
const commonReservedWords = [
  "ABORT", "ABSOLUTE", "ACTION", "ADD", "ALL", "ALTER", "AND", "ANY", "AS",
  "ASC", "BETWEEN", "BY", "CASE", "CAST", "COLUMN", "CONNECT", "COUNT",
  "CREATE", "CURRENT", "DATE", "DELETE", "DESC", "DROP", "ELSE", "EXISTS",
  "FOR", "FROM", "GRANT", "GROUP", "HAVING", "IN", "INDEX", "INSERT", "INTO",
  "IS", "JOIN", "KEY", "LEVEL", "LIKE", "LIMIT", "LOCAL", "MAX", "MIN", "NAME",
  "NOT", "NULL", "OF", "ON", "OR", "ORDER", "OUTER", "REPLACE", "RETURN",
  "SELECT", "SET", "SIZE", "TABLE", "THEN", "TO", "UPDATE", "USER", "VALUES",
  "VIEW", "WHERE"
];
console.log(commonReservedWords.join(", "));

// Explain expression attribute names
console.log("\nKey points about expression attribute names:");
console.log("1. Use expression attribute names (#name) for reserved words");
console.log("2. Use expression attribute names for attributes with special characters");
console.log("3. Special characters include: spaces, hyphens, dots, and other non-alphanumeric characters");
console.log("4. Expression attribute names are required for nested attributes with reserved words");
console.log("5. You can use multiple expression attribute names in a single expression");
console.log("6. Expression attribute names are case-sensitive");
console.log("7. Expression attribute names are only used in expressions, not in the actual data");

} catch (error) {
```

```
    console.error("Error:", error);  
  }  
}
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für JavaScript - API-Referenz.
 - [Query](#)
 - [UpdateItem](#)

Verwendung geplanter Ereignisse zum Aufrufen einer Lambda-Funktion

Das folgende Codebeispiel zeigt, wie eine AWS Lambda Funktion erstellt wird, die durch ein von Amazon EventBridge geplantes Ereignis aufgerufen wird.

SDK für JavaScript (v3)

Zeigt, wie ein von Amazon EventBridge geplantes Ereignis erstellt wird, das eine AWS Lambda Funktion aufruft. Konfigurieren Sie so EventBridge, dass ein Cron-Ausdruck verwendet wird, um zu planen, wann die Lambda-Funktion aufgerufen wird. In diesem Beispiel erstellen Sie eine Lambda-Funktion mithilfe der JavaScript Lambda-Laufzeit-API. In diesem Beispiel werden verschiedene AWS Dienste aufgerufen, um einen bestimmten Anwendungsfall auszuführen. Dieses Beispiel zeigt, wie man eine App erstellt, die eine mobile Textnachricht an Ihre Mitarbeiter sendet, um ihnen zum einjährigen Jubiläum zu gratulieren.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

Dieses Beispiel ist auch verfügbar im [AWS SDK für JavaScript Entwicklerhandbuch für v3](#).

In diesem Beispiel verwendete Dienste

- CloudWatch Logs
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

Serverless-Beispiele

Aufrufen einer Lambda-Funktion über einen DynamoDB-Auslöser

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein durch den Empfang von Datensätzen aus einem DynamoDB-Stream ausgelöstes Ereignis empfängt. Die Funktion ruft die DynamoDB-Nutzdaten ab und protokolliert den Inhalt des Datensatzes.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Ein DynamoDB-Ereignis mit Lambda verwenden. JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  console.log(JSON.stringify(event, null, 2));
  event.Records.forEach(record => {
    logDynamoDBRecord(record);
  });
};

const logDynamoDBRecord = (record) => {
  console.log(record.eventID);
  console.log(record.eventName);
  console.log(`DynamoDB Record: ${JSON.stringify(record.dynamodb)}`);
};
```

Ein DynamoDB-Ereignis mit Lambda verwenden. TypeScript

```
export const handler = async (event, context) => {
  console.log(JSON.stringify(event, null, 2));
  event.Records.forEach(record => {
    logDynamoDBRecord(record);
  });
}
```



```
const logDynamoDBRecord = (record) => {  
  console.log(record.eventID);  
  console.log(record.eventName);  
  console.log(`DynamoDB Record: ${JSON.stringify(record.dynamodb)}`);  
};
```

Melden von Batch-Elementfehlern für Lambda-Funktionen mit einem DynamoDB-Auslöser

Das folgende Codebeispiel zeigt, wie eine teilweise Batch-Antwort für Lambda-Funktionen implementiert wird, die Ereignisse von einem DynamoDB-Stream empfangen. Die Funktion meldet die Batch-Elementfehler in der Antwort und signalisiert Lambda, diese Nachrichten später erneut zu versuchen.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Melden von DynamoDB-Batchelementfehlern mit Lambda unter Verwendung von JavaScript

```
export const handler = async (event) => {  
  const records = event.Records;  
  let curRecordSequenceNumber = "";  
  
  for (const record of records) {  
    try {  
      // Process your record  
      curRecordSequenceNumber = record.dynamodb.SequenceNumber;  
    } catch (e) {  
      // Return failed record's sequence number  
      return { batchItemFailures: [{ itemIdentifier: curRecordSequenceNumber }] };  
    }  
  }  
  
  return { batchItemFailures: [] };  
};
```

Melden von DynamoDB-Batchelementfehlern mit Lambda unter Verwendung von. TypeScript

```
import {
  DynamoDBBatchResponse,
  DynamoDBBatchItemFailure,
  DynamoDBStreamEvent,
} from "aws-lambda";

export const handler = async (
  event: DynamoDBStreamEvent
): Promise<DynamoDBBatchResponse> => {
  const batchItemFailures: DynamoDBBatchItemFailure[] = [];
  let curRecordSequenceNumber;

  for (const record of event.Records) {
    curRecordSequenceNumber = record.dynamodb?.SequenceNumber;

    if (curRecordSequenceNumber) {
      batchItemFailures.push({
        itemIdentifier: curRecordSequenceNumber,
      });
    }
  }

  return { batchItemFailures: batchItemFailures };
};
```

EC2 Amazon-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK für JavaScript (v3) mit Amazon Aktionen ausführen und allgemeine Szenarien implementieren EC2.

Bei Grundlagen handelt es sich um Codebeispiele, die Ihnen zeigen, wie Sie die wesentlichen Vorgänge innerhalb eines Services ausführen.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien anzeigen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb eines Service aufrufen oder mit anderen AWS-Services kombinieren.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kodex finden.

Themen

- [Erste Schritte](#)
- [Grundlagen](#)
- [Aktionen](#)
- [Szenarien](#)

Erste Schritte

Hallo Amazon EC2

Das folgende Codebeispiel zeigt, wie Sie mit Amazon beginnen können EC2.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { DescribeSecurityGroupsCommand, EC2Client } from "@aws-sdk/client-ec2";

// Call DescribeSecurityGroups and display the result.
export const main = async () => {
  const client = new EC2Client();
  try {
    const { SecurityGroups } = await client.send(
      new DescribeSecurityGroupsCommand({}),
    );

    const securityGroupList = SecurityGroups.slice(0, 9)
```

```
    .map((sg) => ` • ${sg.GroupId}: ${sg.GroupName}`)
    .join("\n");

    console.log(
      "Hello, Amazon EC2! Let's list up to 10 of your security groups:",
    );
    console.log(securityGroupList);
  } catch (err) {
    console.error(err);
  }
};

// Call function if run directly.
import { fileURLToPath } from "node:url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main();
}
```

- Einzelheiten zur API finden Sie [DescribeSecurityGroups](#) in der AWS SDK für JavaScript API-Referenz.

Grundlagen

Kennenlernen der Grundlagen

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Erstellen Sie ein Schlüsselpaar und eine Sicherheitsgruppe.
- Wählen Sie ein Amazon Machine Image (AMI) und einen kompatiblen Instance-Typ aus und erstellen Sie anschließend eine Instance.
- Halten Sie die Instance an und starten Sie sie neu.
- Verknüpfen einer Elastic-IP-Adresse mit der Instance.
- Stellen Sie über SSH eine Verbindung zu Ihrer Instance her und bereinigen Sie dann die Ressourcen.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Diese Datei enthält eine Liste gängiger Aktionen, die mit verwendet werden EC2. Die Schritte sind mit einem Szenario-Framework aufgebaut, das die Ausführung eines interaktiven Beispiels vereinfacht. Den vollständigen Kontext finden Sie im GitHub Repository.

```
import { tmpdir } from "node:os";
import { writeFile, mkdtemp, rm } from "node:fs/promises";
import { join } from "node:path";
import { get } from "node:http";

import {
  AllocateAddressCommand,
  AssociateAddressCommand,
  AuthorizeSecurityGroupIngressCommand,
  CreateKeyPairCommand,
  CreateSecurityGroupCommand,
  DeleteKeyPairCommand,
  DeleteSecurityGroupCommand,
  DisassociateAddressCommand,
  paginateDescribeImages,
  paginateDescribeInstances,
  paginateDescribeInstanceTypes,
  ReleaseAddressCommand,
  RunInstancesCommand,
  StartInstancesCommand,
  StopInstancesCommand,
  TerminateInstancesCommand,
  waitUntilInstanceStatusOk,
  waitUntilInstanceStopped,
  waitUntilInstanceTerminated,
} from "@aws-sdk/client-ec2";

import {
  ScenarioAction,
  ScenarioInput,
```

```

    ScenarioOutput,
  } from "@aws-doc-sdk-examples/lib/scenario/index.js";

import { paginateGetParametersByPath, SSMClient } from "@aws-sdk/client-ssm";

/**
 * @typedef {{
 *   ec2Client: import('@aws-sdk/client-ec2').EC2Client,
 *   errors: Error[],
 *   keyPairId?: string,
 *   tmpDirectory?: string,
 *   securityGroupId?: string,
 *   ipAddress?: string,
 *   images?: import('@aws-sdk/client-ec2').Image[],
 *   image?: import('@aws-sdk/client-ec2').Image,
 *   instanceTypes?: import('@aws-sdk/client-ec2').InstanceTypeInfo[],
 *   instanceId?: string,
 *   instanceIpAddress?: string,
 *   allocationId?: string,
 *   allocatedIpAddress?: string,
 *   associationId?: string,
 * }} State
 */

/**
 * A skip function provided to the `skipWhen` of a Step when you want
 * to ignore that step if any errors have occurred.
 * @param {State} state
 */
const skipWhenErrors = (state) => state.errors.length > 0;

const MAX_WAITER_TIME_IN_SECONDS = 60 * 8;

export const confirm = new ScenarioInput("confirmContinue", "Continue?", {
  type: "confirm",
  skipWhen: skipWhenErrors,
});

export const exitOnNoConfirm = new ScenarioAction(
  "exitOnConfirmContinueFalse",
  (/** @type { { earlyExit: boolean } & Record<string, any> } */ state) => {
    if (!state[confirm.name]) {
      state.earlyExit = true;
    }
  })

```

```

    },
    {
      skipWhen: skipWhenErrors,
    },
  );

```

```

export const greeting = new ScenarioOutput(
  "greeting",
  `

```

Welcome to the Amazon EC2 basic usage scenario.

Before you launch an instances, you'll need to provide a few things:

- A key pair - This is for SSH access to your EC2 instance. You only need to provide the name.
- A security group - This is used for configuring access to your instance. Again, only the name is needed.
- An IP address - Your public IP address will be fetched.
- An Amazon Machine Image (AMI)
- A compatible instance type`,

```

  { header: true, preformatted: true, skipWhen: skipWhenErrors },
);

```

```

export const provideKeyName = new ScenarioInput(
  "keyPairName",
  "Provide a name for a new key pair.",
  { type: "input", default: "ec2-example-key-pair", skipWhen: skipWhenErrors },
);

```

```

export const createKeyPair = new ScenarioAction(
  "createKeyPair",
  async (** @type {State} */ state) => {
    try {
      // Create a key pair in Amazon EC2.
      const { KeyMaterial, KeyPairId } = await state.ec2Client.send(
        // A unique name for the key pair. Up to 255 ASCII characters.
        new CreateKeyPairCommand({ KeyName: state[provideKeyName.name] }),
      );

      state.keyPairId = KeyPairId;

      // Save the private key in a temporary location.
      state.tmpDirectory = await mkdtemp(join(tmpdir(), "ec2-scenario-tmp"));
      await writeFile(

```

```

        `${state.tmpDirectory}/${state[provideKeyPairName.name]}.pem`,
        KeyMaterial,
        {
            mode: 0o400,
        },
    );
} catch (caught) {
    if (
        caught instanceof Error &&
        caught.name === "InvalidKeyPair.Duplicate"
    ) {
        caught.message = `${caught.message}. Try another key name.`;
    }

    state.errors.push(caught);
}
},
{ skipWhen: skipWhenErrors },
);

export const logKeyPair = new ScenarioOutput(
    "logKeyPair",
    (/** @type {State} */ state) =>
        `Created the key pair ${state[provideKeyPairName.name]}.`,
    { skipWhen: skipWhenErrors },
);

export const confirmDeleteKeyPair = new ScenarioInput(
    "confirmDeleteKeyPair",
    "Do you want to delete the key pair?",
    {
        type: "confirm",
        // Don't do anything when a key pair was never created.
        skipWhen: (/** @type {State} */ state) => !state.keyPairId,
    },
);

export const maybeDeleteKeyPair = new ScenarioAction(
    "deleteKeyPair",
    async (/** @type {State} */ state) => {
        try {
            // Delete a key pair by name from EC2
            await state.ec2Client.send(
                new DeleteKeyPairCommand({ KeyName: state[provideKeyPairName.name] }),
            );
        } catch (error) {
            state.errors.push(error);
        }
    },
    { skipWhen: skipWhenErrors },
);

```



```

    );
  } catch (caught) {
    if (
      caught instanceof Error &&
      // Occurs when a required parameter (e.g. KeyName) is undefined.
      caught.name === "MissingParameter"
    ) {
      caught.message = `${caught.message}. Did you provide the required value?`;
    }
    state.errors.push(caught);
  }
},
{
  // Don't do anything when there's no key pair to delete or the user chooses
  // to keep it.
  skipWhen: (/** @type {State} */ state) =>
    !state.keyPairId || !state[confirmDeleteKeyPair.name],
},
);

export const provideSecurityGroupName = new ScenarioInput(
  "securityGroupName",
  "Provide a name for a new security group.",
  { type: "input", default: "ec2-scenario-sg", skipWhen: skipWhenErrors },
);

export const createSecurityGroup = new ScenarioAction(
  "createSecurityGroup",
  async (/** @type {State} */ state) => {
    try {
      // Create a new security group that will be used to configure ingress/egress
      for
      // an EC2 instance.
      const { GroupId } = await state.ec2Client.send(
        new CreateSecurityGroupCommand({
          GroupName: state[provideSecurityGroupName.name],
          Description: "A security group for the Amazon EC2 example.",
        }),
      );
      state.securityGroupId = GroupId;
    } catch (caught) {
      if (caught instanceof Error && caught.name === "InvalidGroup.Duplicate") {
        caught.message = `${caught.message}. Please provide a different name for
        your security group.`;
      }
    }
  }
);

```

```
    }

    state.errors.push(caught);
  }
},
{ skipWhen: skipWhenErrors },
);

export const logSecurityGroup = new ScenarioOutput(
  "logSecurityGroup",
  (** @type {State} */ state) =>
    `Created the security group ${state.securityGroupId}.`,
  { skipWhen: skipWhenErrors },
);

export const confirmDeleteSecurityGroup = new ScenarioInput(
  "confirmDeleteSecurityGroup",
  "Do you want to delete the security group?",
  {
    type: "confirm",
    // Don't do anything when a security group was never created.
    skipWhen: (** @type {State} */ state) => !state.securityGroupId,
  },
);

export const maybeDeleteSecurityGroup = new ScenarioAction(
  "deleteSecurityGroup",
  async (** @type {State} */ state) => {
    try {
      // Delete the security group if the 'skipWhen' condition below is not met.
      await state.ec2Client.send(
        new DeleteSecurityGroupCommand({
          GroupId: state.securityGroupId,
        }),
      );
    } catch (caught) {
      if (
        caught instanceof Error &&
        caught.name === "InvalidGroupId.Malformed"
      ) {
        caught.message = `${caught.message}. Please provide a valid GroupId.`;
      }
      state.errors.push(caught);
    }
  }
);
```

```

    },
    {
      // Don't do anything when there's no security group to delete
      // or the user chooses to keep it.
      skipWhen: (** @type {State} */ state) =>
        !state.securityGroupId || !state[confirmDeleteSecurityGroup.name],
    },
  );

export const authorizeSecurityGroupIngress = new ScenarioAction(
  "authorizeSecurity",
  async (** @type {State} */ state) => {
    try {
      // Get the public IP address of the machine running this example.
      const ipAddress = await new Promise((res, rej) => {
        get("http://checkip.amazonaws.com", (response) => {
          let data = "";
          response.on("data", (chunk) => {
            data += chunk;
          });
          response.on("end", () => res(data.trim()));
        }).on("error", (err) => {
          rej(err);
        });
      });
      state.ipAddress = ipAddress;
      // Allow ingress from the IP address above to the security group.
      // This will allow you to SSH into the EC2 instance.
      const command = new AuthorizeSecurityGroupIngressCommand({
        GroupId: state.securityGroupId,
        IpPermissions: [
          {
            IpProtocol: "tcp",
            FromPort: 22,
            ToPort: 22,
            IpRanges: [{ CidrIp: `${ipAddress}/32` }],
          },
        ],
      });

      await state.ec2Client.send(command);
    } catch (caught) {
      if (
        caught instanceof Error &&

```

```

        caught.name === "InvalidGroupId.Malformed"
    ) {
        caught.message = `${caught.message}. Please provide a valid GroupId.`;
    }

    state.errors.push(caught);
}
},
{ skipWhen: skipWhenErrors },
);

export const logSecurityGroupIngress = new ScenarioOutput(
    "logSecurityGroupIngress",
    (** @type {State} */ state) =>
    `Allowed SSH access from your public IP: ${state.ipAddress}.`,
    { skipWhen: skipWhenErrors },
);

export const getImages = new ScenarioAction(
    "images",
    async (** @type {State} */ state) => {
        const AMIs = [];
        // Some AWS services publish information about common artifacts as AWS Systems
        // Manager (SSM)
        // public parameters. For example, the Amazon Elastic Compute Cloud (Amazon EC2)
        // service publishes information about Amazon Machine Images (AMIs) as public
        // parameters.

        // Create the paginator for getting images. Actions that return multiple pages
        // of
        // results have paginators to simplify those calls.
        const getParametersByPathPaginator = paginateGetParametersByPath(
            {
                // Not storing this client in state since it's only used once.
                client: new SSMClient({}),
            },
            {
                // The path to the public list of the latest amazon-linux instances.
                Path: "/aws/service/ami-amazon-linux-latest",
            },
        );

        try {
            for await (const page of getParametersByPathPaginator) {

```

```
    for (const param of page.Parameters) {
      // Filter by Amazon Linux 2
      if (param.Name.includes("amzn2")) {
        AMIs.push(param.Value);
      }
    }
  } catch (caught) {
    if (caught instanceof Error && caught.name === "InvalidFilterValue") {
      caught.message = `${caught.message} Please provide a valid filter value for
paginateGetParametersByPath.`;
    }
    state.errors.push(caught);
    return;
  }

  const imageDetails = [];
  const describeImagesPaginator = paginateDescribeImages(
    { client: state.ec2Client },
    // The images found from the call to SSM.
    { ImageIds: AMIs },
  );

  try {
    // Get more details for the images found above.
    for await (const page of describeImagesPaginator) {
      imageDetails.push(...(page.Images || []));
    }

    // Store the image details for later use.
    state.images = imageDetails;
  } catch (caught) {
    if (caught instanceof Error && caught.name === "InvalidAMIID.NotFound") {
      caught.message = `${caught.message}. Please provide a valid image id.`;
    }

    state.errors.push(caught);
  }
},
{ skipWhen: skipWhenErrors },
);

export const provideImage = new ScenarioInput(
  "image",
```

```

    "Select one of the following images.",
    {
      type: "select",
      choices: (/** @type { State } */ state) =>
        state.images.map((image) => ({
          name: `${image.Description}`,
          value: image,
        })),
      default: (/** @type { State } */ state) => state.images[0],
      skipWhen: skipWhenErrors,
    },
  );

export const getCompatibleInstanceTypes = new ScenarioAction(
  "getCompatibleInstanceTypes",
  async (/** @type {State} */ state) => {
    // Get more details about instance types that match the architecture of
    // the provided image.
    const paginator = paginateDescribeInstanceTypes(
      { client: state.ec2Client, pageSize: 25 },
      {
        Filters: [
          {
            Name: "processor-info.supported-architecture",
            // The value selected from provideImage()
            Values: [state.image.Architecture],
          },
          // Filter for smaller, less expensive, types.
          { Name: "instance-type", Values: ["*.micro", "/*.small"] },
        ],
      },
    );

    const instanceTypes = [];

    try {
      for await (const page of paginator) {
        if (page.InstanceTypes.length) {
          instanceTypes.push(...(page.InstanceTypes || []));
        }
      }

      if (!instanceTypes.length) {
        state.errors.push(

```

```

        "No instance types matched the instance type filters.",
    );
}
} catch (caught) {
    if (caught instanceof Error && caught.name === "InvalidParameterValue") {
        caught.message = `${caught.message}. Please check the provided values and
try again.`;
    }

    state.errors.push(caught);
}

state.instanceTypes = instanceTypes;
},
{ skipWhen: skipWhenErrors },
);

export const provideInstanceType = new ScenarioInput(
    "instanceType",
    "Select an instance type.",
    {
        choices: (/** @type {State} */ state) =>
            state.instanceTypes.map((instanceType) => ({
                name: `${instanceType.InstanceType} - Memory:
${instanceType.MemoryInfo.SizeInMiB}`,
                value: instanceType.InstanceType,
            })),
        type: "select",
        default: (/** @type {State} */ state) =>
            state.instanceTypes[0].InstanceType,
        skipWhen: skipWhenErrors,
    },
);

export const runInstance = new ScenarioAction(
    "runInstance",
    async (/** @type { State } */ state) => {
        const { Instances } = await state.ec2Client.send(
            new RunInstancesCommand({
                KeyName: state[provideKeyPairName.name],
                SecurityGroupIds: [state.securityGroupId],
                ImageId: state.image.ImageId,
                InstanceType: state[provideInstanceType.name],
            })
        );
    }
);

```

```

    // Availability Zones have capacity limitations that may impact your ability
    to launch instances.
    // The `RunInstances` operation will only succeed if it can allocate at
    least the `MinCount` of instances.
    // However, EC2 will attempt to launch up to the `MaxCount` of instances,
    even if the full request cannot be satisfied.
    // If you need a specific number of instances, use `MinCount` and `MaxCount`
    set to the same value.
    // If you want to launch up to a certain number of instances, use `MaxCount`
    and let EC2 provision as many as possible.
    // If you require a minimum number of instances, but do not want to exceed a
    maximum, use both `MinCount` and `MaxCount`.
    MinCount: 1,
    MaxCount: 1,
  }},
);

state.instanceId = Instances[0].InstanceId;

try {
  // Poll `DescribeInstanceStatus` until status is "ok".
  await waitUntilInstanceStatusOk(
    {
      client: state.ec2Client,
      maxWaitTime: MAX_WAITER_TIME_IN_SECONDS,
    },
    { InstanceIds: [Instances[0].InstanceId] },
  );
} catch (caught) {
  if (caught instanceof Error && caught.name === "TimeoutError") {
    caught.message = `${caught.message}. Try increasing the maxWaitTime in the
waiter.`;
  }

  state.errors.push(caught);
}
},
{ skipWhen: skipWhenErrors },
);

export const logRunInstance = new ScenarioOutput(
  "logRunInstance",
  "The next step is to run your EC2 instance for the first time. This can take a few
minutes.",

```



```

    { header: true, skipWhen: skipWhenErrors },
  );

export const describeInstance = new ScenarioAction(
  "describeInstance",
  async (/** @type { State } */ state) => {
    /** @type { import("@aws-sdk/client-ec2").Instance[] } */
    const instances = [];

    try {
      const paginator = paginateDescribeInstances(
        {
          client: state.ec2Client,
        },
        {
          // Only get our created instance.
          InstanceIds: [state.instanceId],
        },
      );

      for await (const page of paginator) {
        for (const reservation of page.Reservations) {
          instances.push(...reservation.Instances);
        }
      }
      if (instances.length !== 1) {
        throw new Error(`Instance ${state.instanceId} not found.`);
      }

      // The only info we need is the IP address for SSH purposes.
      state.instanceIpAddress = instances[0].PublicIpAddress;
    } catch (caught) {
      if (caught instanceof Error && caught.name === "InvalidParameterValue") {
        caught.message = `${caught.message}. Please check provided values and try again.`;
      }

      state.errors.push(caught);
    }
  },
  { skipWhen: skipWhenErrors },
);

export const logSSHConnectionInfo = new ScenarioOutput(

```

```
"logSSHConnectionInfo",
  (** @type { State } */ state) =>
    `You can now SSH into your instance using the following command:
ssh -i ${state.tmpDirectory}/${state[provideKeyPairName.name]}.pem ec2-user@
${state.instanceIpAddress}`,
    { preformatted: true, skipWhen: skipWhenErrors },
  );

export const logStopInstance = new ScenarioOutput(
  "logStopInstance",
  "Stopping your EC2 instance.",
  { skipWhen: skipWhenErrors },
);

export const stopInstance = new ScenarioAction(
  "stopInstance",
  async (** @type { State } */ state) => {
    try {
      await state.ec2Client.send(
        new StopInstancesCommand({
          InstanceIds: [state.instanceId],
        }),
      );

      await waitUntilInstanceStopped(
        {
          client: state.ec2Client,
          maxWaitTime: MAX_WAITER_TIME_IN_SECONDS,
        },
        { InstanceIds: [state.instanceId] },
      );
    } catch (caught) {
      if (caught instanceof Error && caught.name === "TimeoutError") {
        caught.message = `${caught.message}. Try increasing the maxWaitTime in the
waiter.`;
      }

      state.errors.push(caught);
    }
  },
  // Don't try to stop an instance that doesn't exist.
  { skipWhen: (** @type { State } */ state) => !state.instanceId },
);
```

```
export const logIpAddressBehavior = new ScenarioOutput(
  "logIpAddressBehavior",
  [
    "When you run an instance, by default it's assigned an IP address.",
    "That IP address is not static. It will change every time the instance is restarted.",
    "The next step is to stop and restart your instance to demonstrate this behavior.",
  ].join(" "),
  { header: true, skipWhen: skipWhenErrors },
);

export const logStartInstance = new ScenarioOutput(
  "logStartInstance",
  (/** @type { State } */ state) => `Starting instance ${state.instanceId}`,
  { skipWhen: skipWhenErrors },
);

export const startInstance = new ScenarioAction(
  "startInstance",
  async (/** @type { State } */ state) => {
    try {
      await state.ec2Client.send(
        new StartInstancesCommand({
          InstanceIds: [state.instanceId],
        }),
      );

      await waitUntilInstanceStatusOk(
        {
          client: state.ec2Client,
          maxWaitTime: MAX_WAITER_TIME_IN_SECONDS,
        },
        { InstanceIds: [state.instanceId] },
      );
    } catch (caught) {
      if (caught instanceof Error && caught.name === "TimeoutError") {
        caught.message = `${caught.message}. Try increasing the maxWaitTime in the waiter.`;
      }

      state.errors.push(caught);
    }
  },
);
```

```
    { skipWhen: skipWhenErrors },
  );

export const logIpAllocation = new ScenarioOutput(
  "logIpAllocation",
  [
    "It is possible to have a static IP address.",
    "To demonstrate this, an IP will be allocated and associated to your EC2 instance.",
  ].join(" "),
  { header: true, skipWhen: skipWhenErrors },
);

export const allocateIp = new ScenarioAction(
  "allocateIp",
  async (** @type { State } */ state) => {
    try {
      // An Elastic IP address is allocated to your AWS account, and is yours until you release it.
      const { AllocationId, PublicIp } = await state.ec2Client.send(
        new AllocateAddressCommand({}),
      );
      state.allocationId = AllocationId;
      state.allocatedIpAddress = PublicIp;
    } catch (caught) {
      if (caught instanceof Error && caught.name === "MissingParameter") {
        caught.message = `${caught.message}. Did you provide these values`;
      }
      state.errors.push(caught);
    }
  },
  { skipWhen: skipWhenErrors },
);

export const associateIp = new ScenarioAction(
  "associateIp",
  async (** @type { State } */ state) => {
    try {
      // Associate an allocated IP address to an EC2 instance. An IP address can be allocated
      // with the AllocateAddress action.
      const { AssociationId } = await state.ec2Client.send(
        new AssociateAddressCommand({
          AllocationId: state.allocationId,
```

```

        InstanceId: state.instanceId,
    )),
    );
    state.associationId = AssociationId;
    // Update the IP address that is being tracked to match
    // the one just associated.
    state.instanceIpAddress = state.allocatedIpAddress;
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "InvalidAllocationID.NotFound"
    ) {
      caught.message = `${caught.message}. Did you provide the ID of a valid
Elastic IP address AllocationId?`;
    }
    state.errors.push(caught);
  }
},
{ skipWhen: skipWhenErrors },
);

export const logStaticIpProof = new ScenarioOutput(
  "logStaticIpProof",
  "The IP address should remain the same even after stopping and starting the
instance.",
  { header: true, skipWhen: skipWhenErrors },
);

export const logCleanUp = new ScenarioOutput(
  "logCleanUp",
  "That's it! You can choose to clean up the resources now, or clean them up on your
own later.",
  { header: true, skipWhen: skipWhenErrors },
);

export const confirmDisassociateAddress = new ScenarioInput(
  "confirmDisassociateAddress",
  "Do you want to disassociate and release the static IP address created earlier?",
  {
    type: "confirm",
    skipWhen: (/** @type { State } */ state) => !state.associationId,
  },
);

```

```

export const maybeDisassociateAddress = new ScenarioAction(
  "maybeDisassociateAddress",
  async (/** @type { State } */ state) => {
    try {
      await state.ec2Client.send(
        new DisassociateAddressCommand({
          AssociationId: state.associationId,
        }),
      );
    } catch (caught) {
      if (
        caught instanceof Error &&
        caught.name === "InvalidAssociationID.NotFound"
      ) {
        caught.message = `${caught.message}. Please provide a valid association
ID.`;
      }
      state.errors.push(caught);
    }
  },
  {
    skipWhen: (/** @type { State } */ state) =>
      !state[confirmDisassociateAddress.name] || !state.associationId,
  },
);

export const maybeReleaseAddress = new ScenarioAction(
  "maybeReleaseAddress",
  async (/** @type { State } */ state) => {
    try {
      await state.ec2Client.send(
        new ReleaseAddressCommand({
          AllocationId: state.allocationId,
        }),
      );
    } catch (caught) {
      if (
        caught instanceof Error &&
        caught.name === "InvalidAllocationID.NotFound"
      ) {
        caught.message = `${caught.message}. Please provide a valid AllocationID.`;
      }
      state.errors.push(caught);
    }
  }
);

```

```

    },
    {
      skipWhen: (/** @type { State } */ state) =>
        !state[confirmDisassociateAddress.name] || !state.allocationId,
    },
  );

export const confirmTerminateInstance = new ScenarioInput(
  "confirmTerminateInstance",
  "Do you want to terminate the instance?",
  // Don't do anything when an instance was never run.
  {
    skipWhen: (/** @type { State } */ state) => !state.instanceId,
    type: "confirm",
  },
);

export const maybeTerminateInstance = new ScenarioAction(
  "terminateInstance",
  async (/** @type { State } */ state) => {
    try {
      await state.ec2Client.send(
        new TerminateInstancesCommand({
          InstanceIds: [state.instanceId],
        }),
      );
      await waitUntilInstanceTerminated(
        { client: state.ec2Client },
        { InstanceIds: [state.instanceId] },
      );
    } catch (caught) {
      if (caught instanceof Error && caught.name === "TimeoutError") {
        caught.message = `${caught.message}. Try increasing the maxWaitTime in the
waiter.`;
      }

      state.errors.push(caught);
    }
  },
  {
    // Don't do anything when there's no instance to terminate or the
    // user chooses not to terminate.
    skipWhen: (/** @type { State } */ state) =>
      !state.instanceId || !state[confirmTerminateInstance.name],
  },
);

```

```
    },
  );

export const deleteTemporaryDirectory = new ScenarioAction(
  "deleteTemporaryDirectory",
  async (/** @type { State } */ state) => {
    try {
      await rm(state.tmpDirectory, { recursive: true });
    } catch (caught) {
      state.errors.push(caught);
    }
  },
);

export const logErrors = new ScenarioOutput(
  "logErrors",
  (/** @type {State} */ state) => {
    const errorList = state.errors
      .map((err) => ` - ${err.name}: ${err.message}`)
      .join("\n");
    return `Scenario errors found:\n${errorList}`;
  },
  {
    preformatted: true,
    header: true,
    // Don't log errors when there aren't any!
    skipWhen: (/** @type {State} */ state) => state.errors.length === 0,
  },
);
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für JavaScript - API-Referenz.
 - [AllocateAddress](#)
 - [AssociateAddress](#)
 - [AuthorizeSecurityGroupIngress](#)
 - [CreateKeyPair](#)
 - [CreateSecurityGroup](#)
 - [DeleteKeyPair](#)
 - [DeleteSecurityGroup](#)

- [DescribeImages](#)
- [DescribeInstanceTypes](#)
- [DescribeInstances](#)
- [DescribeKeyPairs](#)
- [DescribeSecurityGroups](#)
- [DisassociateAddress](#)
- [ReleaseAddress](#)
- [RunInstances](#)
- [StartInstances](#)
- [StopInstances](#)
- [TerminateInstances](#)
- [UnmonitorInstances](#)

Aktionen

AllocateAddress

Das folgende Codebeispiel zeigt die Verwendung `AllocateAddress`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { AllocateAddressCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
 * Allocates an Elastic IP address to your AWS account.
 */
export const main = async () => {
  const client = new EC2Client({});
  const command = new AllocateAddressCommand({});
```

```

try {
  const { AllocationId, PublicIp } = await client.send(command);
  console.log("A new IP address has been allocated to your account:");
  console.log(`ID: ${AllocationId} Public IP: ${PublicIp}`);
  console.log(
    "You can view your IP addresses in the AWS Management Console for Amazon EC2. Look under Network & Security > Elastic IPs",
  );
} catch (caught) {
  if (caught instanceof Error && caught.name === "MissingParameter") {
    console.warn(`${caught.message}. Did you provide these values?`);
  } else {
    throw caught;
  }
}
};
import { fileURLToPath } from "node:url";
// Call function if run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main();
}

```

- Einzelheiten zur API finden Sie [AllocateAddress](#) in der AWS SDK für JavaScript API-Referenz.

AssociateAddress

Das folgende Codebeispiel zeigt die Verwendung `AssociateAddress`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```

import { AssociateAddressCommand, EC2Client } from "@aws-sdk/client-ec2";

/**

```

```

* Associates an Elastic IP address, or carrier IP address (for instances that are
in subnets in Wavelength Zones)
* with an instance or a network interface.
* @param {{ instanceId: string, allocationId: string }} options
*/
export const main = async ({ instanceId, allocationId }) => {
  const client = new EC2Client({});
  const command = new AssociateAddressCommand({
    // You need to allocate an Elastic IP address before associating it with an
instance.
    // You can do that with the AllocateAddressCommand.
    AllocationId: allocationId,
    // You need to create an EC2 instance before an IP address can be associated
with it.
    // You can do that with the RunInstancesCommand.
    InstanceId: instanceId,
  });

  try {
    const { AssociationId } = await client.send(command);
    console.log(
      `Address with allocation ID ${allocationId} is now associated with instance
${instanceId}.`,
      `The association ID is ${AssociationId}.`,
    );
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "InvalidAllocationID.NotFound"
    ) {
      console.warn(
        `${caught.message}. Did you provide the ID of a valid Elastic IP address
AllocationId?`,
      );
    } else {
      throw caught;
    }
  }
};

```

- Einzelheiten zur API finden Sie [AssociateAddress](#) in der AWS SDK für JavaScript API-Referenz.

AuthorizeSecurityGroupIngress

Das folgende Codebeispiel zeigt die Verwendung `AuthorizeSecurityGroupIngress`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import {
  AuthorizeSecurityGroupIngressCommand,
  EC2Client,
} from "@aws-sdk/client-ec2";

/**
 * Adds the specified inbound (ingress) rules to a security group.
 * @param {{ groupId: string, ipAddress: string }} options
 */
export const main = async ({ groupId, ipAddress }) => {
  const client = new EC2Client({});
  const command = new AuthorizeSecurityGroupIngressCommand({
    // Use a group ID from the AWS console or
    // the DescribeSecurityGroupsCommand.
    GroupId: groupId,
    IpPermissions: [
      {
        IpProtocol: "tcp",
        FromPort: 22,
        ToPort: 22,
        // The IP address to authorize.
        // For more information on this notation, see
        // https://en.wikipedia.org/wiki/Classless_Inter-
        Domain_Routing#CIDR_notation
        IpRanges: [{ CidrIp: `${ipAddress}/32` }],
      },
    ],
  });

  try {
    const { SecurityGroupRules } = await client.send(command);
```

```

    console.log(JSON.stringify(SecurityGroupRules, null, 2));
  } catch (caught) {
    if (caught instanceof Error && caught.name === "InvalidGroupId.Malformed") {
      console.warn(`${caught.message}. Please provide a valid GroupId.`);
    } else {
      throw caught;
    }
  }
};

```

- Einzelheiten zur API finden Sie [AuthorizeSecurityGroupIngress](#) in der AWS SDK für JavaScript API-Referenz.

CreateKeyPair

Das folgende Codebeispiel zeigt die Verwendung `CreateKeyPair`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```

import { CreateKeyPairCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
 * Creates an ED25519 or 2048-bit RSA key pair with the specified name and in the
 * specified PEM or PPK format.
 * Amazon EC2 stores the public key and displays the private key for you to save to
 * a file.
 * @param {{ keyName: string }} options
 */
export const main = async ({ keyName }) => {
  const client = new EC2Client({});
  const command = new CreateKeyPairCommand({
    KeyName: keyName,
  });
};

```

```
try {
  const { KeyMaterial, KeyName } = await client.send(command);
  console.log(KeyName);
  console.log(KeyMaterial);
} catch (caught) {
  if (caught instanceof Error && caught.name === "InvalidKeyPair.Duplicate") {
    console.warn(`${caught.message}. Try another key name.`);
  } else {
    throw caught;
  }
}
};
```

- Einzelheiten zur API finden Sie [CreateKeyPair](#) in der AWS SDK für JavaScript API-Referenz.

CreateLaunchTemplate

Das folgende Codebeispiel zeigt die Verwendung `CreateLaunchTemplate`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
const ssmClient = new SSMClient({});
const { Parameter } = await ssmClient.send(
  new GetParameterCommand({
    Name: "/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2",
  }),
);
const ec2Client = new EC2Client({});
await ec2Client.send(
  new CreateLaunchTemplateCommand({
    LaunchTemplateName: NAMES.launchTemplateName,
    LaunchTemplateData: {
      InstanceType: "t3.micro",
      ImageId: Parameter.Value,
      IamInstanceProfile: { Name: NAMES.instanceProfileName },
    },
  })
);
```

```

        UserData: readFileSync(
            join(RESOURCES_PATH, "server_startup_script.sh"),
        ).toString("base64"),
        KeyName: NAMES.keyPairName,
    },
}),

```

- Einzelheiten zur API finden Sie [CreateLaunchTemplate](#) in der AWS SDK für JavaScript API-Referenz.

CreateSecurityGroup

Das folgende Codebeispiel zeigt die Verwendung `CreateSecurityGroup`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```

import { CreateSecurityGroupCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
 * Creates a security group.
 * @param {{ groupName: string, description: string }} options
 */
export const main = async ({ groupName, description }) => {
    const client = new EC2Client({});
    const command = new CreateSecurityGroupCommand({
        // Up to 255 characters in length. Cannot start with sg-.
        GroupName: groupName,
        // Up to 255 characters in length.
        Description: description,
    });

    try {
        const { GroupId } = await client.send(command);
        console.log(GroupId);
    }
}

```

```

    } catch (caught) {
      if (caught instanceof Error && caught.name === "InvalidParameterValue") {
        console.warn(`${caught.message}.`);
      } else {
        throw caught;
      }
    }
  }
};

```

- Einzelheiten zur API finden Sie [CreateSecurityGroup](#) in der AWS SDK für JavaScript API-Referenz.

DeleteKeyPair

Das folgende Codebeispiel zeigt die Verwendung `DeleteKeyPair`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```

import { DeleteKeyPairCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
 * Deletes the specified key pair, by removing the public key from Amazon EC2.
 * @param {{ keyName: string }} options
 */
export const main = async ({ keyName }) => {
  const client = new EC2Client({});
  const command = new DeleteKeyPairCommand({
    KeyName: keyName,
  });

  try {
    await client.send(command);
    console.log("Successfully deleted key pair.");
  } catch (caught) {

```



```
    if (caught instanceof Error && caught.name === "MissingParameter") {
      console.warn(`${caught.message}. Did you provide the required value?`);
    } else {
      throw caught;
    }
  }
};
```

- Einzelheiten zur API finden Sie [DeleteKeyPair](#) in der AWS SDK für JavaScript API-Referenz.

DeleteLaunchTemplate

Das folgende Codebeispiel zeigt die Verwendung `DeleteLaunchTemplate`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
await client.send(
  new DeleteLaunchTemplateCommand({
    LaunchTemplateName: NAMES.launchTemplateName,
  }),
);
```

- Einzelheiten zur API finden Sie [DeleteLaunchTemplate](#) in der AWS SDK für JavaScript API-Referenz.

DeleteSecurityGroup

Das folgende Codebeispiel zeigt die Verwendung `DeleteSecurityGroup`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { DeleteSecurityGroupCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
 * Deletes a security group.
 * @param {{ groupId: string }} options
 */
export const main = async ({ groupId }) => {
  const client = new EC2Client({});
  const command = new DeleteSecurityGroupCommand({
    GroupId: groupId,
  });

  try {
    await client.send(command);
    console.log("Security group deleted successfully.");
  } catch (caught) {
    if (caught instanceof Error && caught.name === "InvalidGroupId.Malformed") {
      console.warn(`${caught.message}. Please provide a valid GroupId.`);
    } else {
      throw caught;
    }
  }
};
```

- Einzelheiten zur API finden Sie [DeleteSecurityGroup](#) in der AWS SDK für JavaScript API-Referenz.

DescribeAddresses

Das folgende Codebeispiel zeigt die Verwendung `DescribeAddresses`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { DescribeAddressesCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
 * Describes the specified Elastic IP addresses or all of your Elastic IP addresses.
 * @param {{ allocationId: string }} options
 */
export const main = async ({ allocationId }) => {
  const client = new EC2Client({});
  const command = new DescribeAddressesCommand({
    // You can omit this property to show all addresses.
    AllocationIds: [allocationId],
  });

  try {
    const { Addresses } = await client.send(command);
    const addressList = Addresses.map((address) => ` • ${address.PublicIp}`);
    console.log("Elastic IP addresses:");
    console.log(addressList.join("\n"));
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "InvalidAllocationID.NotFound"
    ) {
      console.warn(`${caught.message}. Please provide a valid AllocationId.`);
    } else {
      throw caught;
    }
  }
};
```

- Einzelheiten zur API finden Sie [DescribeAddresses](#) in der AWS SDK für JavaScript API-Referenz.

DescribeIamInstanceProfileAssociations

Das folgende Codebeispiel zeigt die Verwendung `DescribeIamInstanceProfileAssociations`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
const ec2Client = new EC2Client({});
const { IamInstanceProfileAssociations } = await ec2Client.send(
  new DescribeIamInstanceProfileAssociationsCommand({
    Filters: [
      { Name: "instance-id", Values: [state.targetInstance.InstanceId] },
    ],
  }),
);
```

- Einzelheiten zur API finden Sie [DescribeIamInstanceProfileAssociations](#) in der AWS SDK für JavaScript API-Referenz.

DescribeImages

Das folgende Codebeispiel zeigt die Verwendung `DescribeImages`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { EC2Client, paginateDescribeImages } from "@aws-sdk/client-ec2";

/**
```

```

* Describes the specified images (AMIs, AKIs, and ARIs) available to you or all of
the images available to you.
* @param {{ architecture: string, pageSize: number }} options
*/
export const main = async ({ architecture, pageSize }) => {
  pageSize = Number.parseInt(pageSize);
  const client = new EC2Client({});

  // The paginate function is a wrapper around the base command.
  const paginator = paginateDescribeImages(
    // Without limiting the page size, this call can take a long time. pageSize is
    just sugar for
    // the MaxResults property in the base command.
    { client, pageSize },
    {
      // There are almost 70,000 images available. Be specific with your filtering
      // to increase efficiency.
      // See https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/clients/client-
      ec2/interfaces/describeimagescommandinput.html#filters
      Filters: [{ Name: "architecture", Values: [architecture] }],
    },
  );

  /**
   * @type {import('@aws-sdk/client-ec2').Image[]}
   */
  const images = [];
  let recordsScanned = 0;

  try {
    for await (const page of paginator) {
      recordsScanned += pageSize;
      if (page.Images.length) {
        images.push(...page.Images);
        break;
      }
      console.log(
        `No matching image found yet. Searched ${recordsScanned} records.`,
      );
    }

    if (images.length) {
      console.log(

```

```

        `Found ${images.length} images:\n\n${images.map((image) =>
image.Name).join("\n")}\n`,
    );
    } else {
        console.log(
            `No matching images found. Searched ${recordsScanned} records.\n`,
        );
    }

    return images;
} catch (caught) {
    if (caught instanceof Error && caught.name === "InvalidParameterValue") {
        console.warn(`${caught.message}`);
        return [];
    }
    throw caught;
}
};

```

- Einzelheiten zur API finden Sie [DescribeImages](#) in der AWS SDK für JavaScript API-Referenz.

DescribeInstanceTypes

Das folgende Codebeispiel zeigt die Verwendung `DescribeInstanceTypes`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```

import { EC2Client, paginateDescribeInstanceTypes } from "@aws-sdk/client-ec2";

/**
 * Describes the specified instance types. By default, all instance types for the
 * current Region are described. Alternatively, you can filter the results.
 * @param {{ pageSize: string, supportedArch: string[], freeTier: boolean }} options
 */
export const main = async ({ pageSize, supportedArch, freeTier }) => {

```

```

pageSize = Number.parseInt(pageSize);
const client = new EC2Client({});

// The paginate function is a wrapper around the underlying command.
const paginator = paginateDescribeInstanceTypes(
  // Without limiting the page size, this call can take a long time. pageSize is
  just sugar for
  // the MaxResults property in the underlying command.
  { client, pageSize },
  {
    Filters: [
      {
        Name: "processor-info.supported-architecture",
        Values: supportedArch,
      },
      { Name: "free-tier-eligible", Values: [freeTier ? "true" : "false"] },
    ],
  },
);

try {
  /**
   * @type {import('@aws-sdk/client-ec2').InstanceTypeInfo[]}
   */
  const instanceTypes = [];

  for await (const page of paginator) {
    if (page.InstanceTypes.length) {
      instanceTypes.push(...page.InstanceTypes);

      // When we have at least 1 result, we can stop.
      if (instanceTypes.length >= 1) {
        break;
      }
    }
  }
  console.log(
    `Memory size in MiB for matching instance types:\n\n${instanceTypes.map((it)
=> `${it.InstanceType}: ${it.MemoryInfo.SizeInMiB} MiB`).join("\n")}`,
  );
} catch (caught) {
  if (caught instanceof Error && caught.name === "InvalidParameterValue") {
    console.warn(`${caught.message}`);
    return [];
  }
}

```

```
    }  
    throw caught;  
  }  
};
```

- Einzelheiten zur API finden Sie [DescribeInstanceTypes](#) in der AWS SDK für JavaScript API-Referenz.

DescribeInstances

Das folgende Codebeispiel zeigt die Verwendung `DescribeInstances`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { EC2Client, paginateDescribeInstances } from "@aws-sdk/client-ec2";  
  
/**  
 * List all of your EC2 instances running with the provided architecture that  
 * were launched in the past month.  
 * @param {{ pageSize: string, architectures: string[] }} options  
 */  
export const main = async ({ pageSize, architectures }) => {  
  pageSize = Number.parseInt(pageSize);  
  const client = new EC2Client({});  
  const d = new Date();  
  const year = d.getFullYear();  
  const month = `0${d.getMonth() + 1}`.slice(-2);  
  const launchTimePattern = `${year}-${month}-*`;  
  
  const paginator = paginateDescribeInstances(  
    {  
      client,  
      pageSize,  
    },  
  );
```



```

{
  Filters: [
    { Name: "architecture", Values: architectures },
    { Name: "instance-state-name", Values: ["running"] },
    {
      Name: "launch-time",
      Values: [launchTimePattern],
    },
  ],
},
);

try {
  /**
   * @type {import('@aws-sdk/client-ec2').Instance[]}
   */
  const instanceList = [];
  for await (const page of paginator) {
    const { Reservations } = page;
    for (const reservation of Reservations) {
      instanceList.push(...reservation.Instances);
    }
  }
  console.log(
    `Running instances launched this month:\n\n${instanceList.map((instance) =>
instance.InstanceId).join("\n")}` ,
  );
} catch (caught) {
  if (caught instanceof Error && caught.name === "InvalidParameterValue") {
    console.warn(`${caught.message}.`);
  } else {
    throw caught;
  }
}
};

```

- Einzelheiten zur API finden Sie [DescribeInstances](#) in der AWS SDK für JavaScript API-Referenz.

DescribeKeyPairs

Das folgende Codebeispiel zeigt die Verwendung `DescribeKeyPairs`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { DescribeKeyPairsCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
 * List all key pairs in the current AWS account.
 * @param {{ dryRun: boolean }}
 */
export const main = async ({ dryRun }) => {
  const client = new EC2Client({});
  const command = new DescribeKeyPairsCommand({ DryRun: dryRun });

  try {
    const { KeyPairs } = await client.send(command);
    const keyPairList = KeyPairs.map(
      (kp) => ` • ${kp.KeyPairId}: ${kp.KeyName}`,
    ).join("\n");
    console.log("The following key pairs were found in your account:");
    console.log(keyPairList);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "DryRunOperation") {
      console.log(`${caught.message}`);
    } else {
      throw caught;
    }
  }
};
```

- Einzelheiten zur API finden Sie [DescribeKeyPairs](#) in der AWS SDK für JavaScript API-Referenz.

DescribeRegions

Das folgende Codebeispiel zeigt die Verwendung `DescribeRegions`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { DescribeRegionsCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
 * List all available AWS regions.
 * @param {{ regionNames: string[], includeOptInRegions: boolean }} options
 */
export const main = async ({ regionNames, includeOptInRegions }) => {
  const client = new EC2Client({});
  const command = new DescribeRegionsCommand({
    // By default this command will not show regions that require you to opt-in.
    // When AllRegions is true, even the regions that require opt-in will be
    returned.
    AllRegions: includeOptInRegions,
    // You can omit the Filters property if you want to get all regions.
    Filters: regionNames?.length
      ? [
          {
            Name: "region-name",
            // You can specify multiple values for a filter.
            // You can also use '*' as a wildcard. This will return all
            // of the regions that start with `us-east-`.
            Values: regionNames,
          },
        ]
      : undefined,
  });

  try {
    const { Regions } = await client.send(command);
    const regionsList = Regions.map((reg) => ` • ${reg.RegionName}`);
    console.log("Found regions:");
    console.log(regionsList.join("\n"));
  } catch (caught) {
    if (caught instanceof Error && caught.name === "DryRunOperation") {
```

```

        console.log(`${caught.message}`);
    } else {
        throw caught;
    }
}
};

```

- Einzelheiten zur API finden Sie [DescribeRegions](#) in der AWS SDK für JavaScript API-Referenz.

DescribeSecurityGroups

Das folgende Codebeispiel zeigt die Verwendung `DescribeSecurityGroups`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```

import { DescribeSecurityGroupsCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
 * Describes the specified security groups or all of your security groups.
 * @param {{ groupIds: string[] }} options
 */
export const main = async ({ groupIds = [] }) => {
    const client = new EC2Client({});
    const command = new DescribeSecurityGroupsCommand({
        GroupIds: groupIds,
    });

    try {
        const { SecurityGroups } = await client.send(command);
        const sgList = SecurityGroups.map(
            (sg) => `• ${sg.GroupName} (${sg.GroupId}): ${sg.Description}`,
        ).join("\n");
        if (sgList.length) {
            console.log(`Security groups:\n${sgList}`);
        }
    }
};

```

```
    } else {
      console.log("No security groups found.");
    }
  } catch (caught) {
    if (caught instanceof Error && caught.name === "InvalidGroupId.Malformed") {
      console.warn(`${caught.message}. Please provide a valid GroupId.`);
    } else if (
      caught instanceof Error &&
      caught.name === "InvalidGroup.NotFound"
    ) {
      console.warn(caught.message);
    } else {
      throw caught;
    }
  }
};
```

- Einzelheiten zur API finden Sie [DescribeSecurityGroups](#) in der AWS SDK für JavaScript API-Referenz.

DescribeSubnets

Das folgende Codebeispiel zeigt die Verwendung `DescribeSubnets`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
const client = new EC2Client({});
const { Subnets } = await client.send(
  new DescribeSubnetsCommand({
    Filters: [
      { Name: "vpc-id", Values: [state.defaultVpc] },
      { Name: "availability-zone", Values: state.availabilityZoneNames },
      { Name: "default-for-az", Values: ["true"] },
    ],
  })
);
```

```
    }},  
  );
```

- Einzelheiten zur API finden Sie [DescribeSubnets](#) in der AWS SDK für JavaScript API-Referenz.

DescribeVpcs

Das folgende Codebeispiel zeigt die Verwendung `DescribeVpcs`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
const client = new EC2Client({});  
const { Vpcs } = await client.send(  
  new DescribeVpcsCommand({  
    Filters: [{ Name: "is-default", Values: ["true"] }]},  
  ),  
);
```

- Einzelheiten zur API finden Sie [DescribeVpcs](#) in der AWS SDK für JavaScript API-Referenz.

DisassociateAddress

Das folgende Codebeispiel zeigt die Verwendung `DisassociateAddress`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { DisassociateAddressCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
 * Disassociate an Elastic IP address from an instance.
 * @param {{ associationId: string }} options
 */
export const main = async ({ associationId }) => {
  const client = new EC2Client({});
  const command = new DisassociateAddressCommand({
    // You can also use PublicIp, but that is for EC2 classic which is being
    retired.
    AssociationId: associationId,
  });

  try {
    await client.send(command);
    console.log("Successfully disassociated address");
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "InvalidAssociationID.NotFound"
    ) {
      console.warn(`${caught.message}.`);
    } else {
      throw caught;
    }
  }
};
```

- Einzelheiten zur API finden Sie [DisassociateAddress](#) in der AWS SDK für JavaScript API-Referenz.

MonitorInstances

Das folgende Codebeispiel zeigt die Verwendung `MonitorInstances`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { EC2Client, MonitorInstancesCommand } from "@aws-sdk/client-ec2";

/**
 * Turn on detailed monitoring for the selected instance.
 * By default, metrics are sent to Amazon CloudWatch every 5 minutes.
 * For a cost you can enable detailed monitoring which sends metrics every minute.
 * @param {{ instanceIds: string[] }} options
 */
export const main = async ({ instanceIds }) => {
  const client = new EC2Client({});
  const command = new MonitorInstancesCommand({
    InstanceIds: instanceIds,
  });

  try {
    const { InstanceMonitorings } = await client.send(command);
    const instancesBeingMonitored = InstanceMonitorings.map(
      (im) =>
        ` • Detailed monitoring state for ${im.InstanceId} is
${im.Monitoring.State}.`,
    );
    console.log("Monitoring status:");
    console.log(instancesBeingMonitored.join("\n"));
  } catch (caught) {
    if (caught instanceof Error && caught.name === "InvalidParameterValue") {
      console.warn(`${caught.message}`);
    } else {
      throw caught;
    }
  }
};
```

- Einzelheiten zur API finden Sie [MonitorInstances](#) in der AWS SDK für JavaScript API-Referenz.

RebootInstances

Das folgende Codebeispiel zeigt die Verwendung `RebootInstances`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { EC2Client, RebootInstancesCommand } from "@aws-sdk/client-ec2";

/**
 * Requests a reboot of the specified instances. This operation is asynchronous;
 * it only queues a request to reboot the specified instances.
 * @param {{ instanceIds: string[] }} options
 */
export const main = async ({ instanceIds }) => {
  const client = new EC2Client({});
  const command = new RebootInstancesCommand({
    InstanceIds: instanceIds,
  });

  try {
    await client.send(command);
    console.log("Instance rebooted successfully.");
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "InvalidInstanceID.NotFound"
    ) {
      console.warn(
        `${caught.message}. Please provide the InstanceId of a valid instance to
reboot.`
      );
    } else {
      throw caught;
    }
  }
};
```

- Einzelheiten zur API finden Sie [RebootInstances](#) in der AWS SDK für JavaScript API-Referenz.

ReleaseAddress

Das folgende Codebeispiel zeigt die Verwendung `ReleaseAddress`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { ReleaseAddressCommand, EC2Client } from "@aws-sdk/client-ec2";

/**
 * Release an Elastic IP address.
 * @param {{ allocationId: string }} options
 */
export const main = async ({ allocationId }) => {
  const client = new EC2Client({});
  const command = new ReleaseAddressCommand({
    // You can also use PublicIp, but that is for EC2 classic which is being
    retired.
    AllocationId: allocationId,
  });

  try {
    await client.send(command);
    console.log("Successfully released address.");
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "InvalidAllocationID.NotFound"
    ) {
      console.warn(`${caught.message}. Please provide a valid AllocationID.`);
    } else {
      throw caught;
    }
  }
}
```

```
    }  
  }  
};
```

- Einzelheiten zur API finden Sie [ReleaseAddress](#) in der AWS SDK für JavaScript API-Referenz.

ReplaceIamInstanceProfileAssociation

Das folgende Codebeispiel zeigt die Verwendung `ReplaceIamInstanceProfileAssociation`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>  
  ec2Client.send(  
    new ReplaceIamInstanceProfileAssociationCommand({  
      AssociationId: state.instanceProfileAssociationId,  
      IamInstanceProfile: { Name: NAMES.ssmOnlyInstanceProfileName },  
    }),  
  ),  
);
```

- Einzelheiten zur API finden Sie [ReplaceIamInstanceProfileAssociation](#) in der AWS SDK für JavaScript API-Referenz.

RunInstances

Das folgende Codebeispiel zeigt die Verwendung `RunInstances`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { EC2Client, RunInstancesCommand } from "@aws-sdk/client-ec2";

/**
 * Create new EC2 instances.
 * @param {{
 *   keyName: string,
 *   securityGroupIds: string[],
 *   imageId: string,
 *   instanceType: import('@aws-sdk/client-ec2')._InstanceType,
 *   minCount?: number,
 *   maxCount?: number }} options
 */
export const main = async ({
  keyName,
  securityGroupIds,
  imageId,
  instanceType,
  minCount = "1",
  maxCount = "1",
}) => {
  const client = new EC2Client({});
  minCount = Number.parseInt(minCount);
  maxCount = Number.parseInt(maxCount);
  const command = new RunInstancesCommand({
    // Your key pair name.
    KeyName: keyName,
    // Your security group.
    SecurityGroupIds: securityGroupIds,
    // An Amazon Machine Image (AMI). There are multiple ways to search for AMIs.
    // For more information, see:
    // https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/finding-an-ami.html
    ImageId: imageId,
    // An instance type describing the resources provided to your instance. There
    // are multiple
  });
  const data = await client.send(command);
  console.log("Instances created:", data.InstanceIds);
}
```

```

// ways to search for instance types. For more information see:
// https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instance-discovery.html
InstanceType: instanceType,
// Availability Zones have capacity limitations that may impact your ability to
launch instances.
// The `RunInstances` operation will only succeed if it can allocate at least
the `MinCount` of instances.
// However, EC2 will attempt to launch up to the `MaxCount` of instances, even
if the full request cannot be satisfied.
// If you need a specific number of instances, use `MinCount` and `MaxCount` set
to the same value.
// If you want to launch up to a certain number of instances, use `MaxCount` and
let EC2 provision as many as possible.
// If you require a minimum number of instances, but do not want to exceed a
maximum, use both `MinCount` and `MaxCount`.
MinCount: minCount,
MaxCount: maxCount,
});

try {
  const { Instances } = await client.send(command);
  const instanceList = Instances.map(
    (instance) => `• ${instance.InstanceId}`,
  ).join("\n");
  console.log(`Launched instances:\n${instanceList}`);
} catch (caught) {
  if (caught instanceof Error && caught.name === "ResourceCountExceeded") {
    console.warn(`${caught.message}`);
  } else {
    throw caught;
  }
}
};

```

- Einzelheiten zur API finden Sie [RunInstances](#) in der AWS SDK für JavaScript API-Referenz.

StartInstances

Das folgende Codebeispiel zeigt die Verwendung `StartInstances`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { EC2Client, StartInstancesCommand } from "@aws-sdk/client-ec2";
import { fileURLToPath } from "node:url";
import { parseArgs } from "node:util";

/**
 * Starts an Amazon EBS-backed instance that you've previously stopped.
 * @param {{ instanceIds }} options
 */
export const main = async ({ instanceIds }) => {
  const client = new EC2Client({});
  const command = new StartInstancesCommand({
    InstanceIds: instanceIds,
  });

  try {
    const { StartingInstances } = await client.send(command);
    const instanceIdList = StartingInstances.map(
      (instance) => ` • ${instance.InstanceId}`,
    );
    console.log("Starting instances:");
    console.log(instanceIdList.join("\n"));
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "InvalidInstanceID.NotFound"
    ) {
      console.warn(`${caught.message}`);
    } else {
      throw caught;
    }
  }
};
```

- Einzelheiten zur API finden Sie [StartInstances](#) in der AWS SDK für JavaScript API-Referenz.

StopInstances

Das folgende Codebeispiel zeigt die Verwendung `StopInstances`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { EC2Client, StopInstancesCommand } from "@aws-sdk/client-ec2";
import { fileURLToPath } from "node:url";
import { parseArgs } from "node:util";

/**
 * Stop one or more EC2 instances.
 * @param {{ instanceIds: string[] }} options
 */
export const main = async ({ instanceIds }) => {
  const client = new EC2Client({});
  const command = new StopInstancesCommand({
    InstanceIds: instanceIds,
  });

  try {
    const { StoppingInstances } = await client.send(command);
    const instanceIdList = StoppingInstances.map(
      (instance) => ` • ${instance.InstanceId}`,
    );
    console.log("Stopping instances:");
    console.log(instanceIdList.join("\n"));
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "InvalidInstanceID.NotFound"
    ) {
      console.warn(` ${caught.message} `);
    } else {

```

```
        throw caught;
    }
}
};
```

- Einzelheiten zur API finden Sie [StopInstances](#) in der AWS SDK für JavaScript API-Referenz.

TerminateInstances

Das folgende Codebeispiel zeigt die Verwendung `TerminateInstances`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { EC2Client, TerminateInstancesCommand } from "@aws-sdk/client-ec2";
import { fileURLToPath } from "node:url";
import { parseArgs } from "node:util";

/**
 * Terminate one or more EC2 instances.
 * @param {{ instanceIds: string[] }} options
 */
export const main = async ({ instanceIds }) => {
  const client = new EC2Client({});
  const command = new TerminateInstancesCommand({
    InstanceIds: instanceIds,
  });

  try {
    const { TerminatingInstances } = await client.send(command);
    const instanceList = TerminatingInstances.map(
      (instance) => ` • ${instance.InstanceId}`,
    );
    console.log("Terminating instances:");
    console.log(instanceList.join("\n"));
  }
};
```



```

    } catch (caught) {
      if (
        caught instanceof Error &&
        caught.name === "InvalidInstanceID.NotFound"
      ) {
        console.warn(`${caught.message}`);
      } else {
        throw caught;
      }
    }
  };

```

- Einzelheiten zur API finden Sie [TerminateInstances](#) in der AWS SDK für JavaScript API-Referenz.

UnmonitorInstances

Das folgende Codebeispiel zeigt die Verwendung `UnmonitorInstances`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```

import { EC2Client, UnmonitorInstancesCommand } from "@aws-sdk/client-ec2";
import { fileURLToPath } from "node:url";
import { parseArgs } from "node:util";

/**
 * Turn off detailed monitoring for the selected instance.
 * @param {{ instanceIds: string[] }} options
 */
export const main = async ({ instanceIds }) => {
  const client = new EC2Client({});
  const command = new UnmonitorInstancesCommand({
    InstanceIds: instanceIds,
  });

```

```
try {
  const { InstanceMonitorings } = await client.send(command);
  const instanceMonitoringsList = InstanceMonitorings.map(
    (im) =>
      ` • Detailed monitoring state for ${im.InstanceId} is
${im.Monitoring.State}.`,
  );
  console.log("Monitoring status:");
  console.log(instanceMonitoringsList.join("\n"));
} catch (caught) {
  if (
    caught instanceof Error &&
    caught.name === "InvalidInstanceID.NotFound"
  ) {
    console.warn(`${caught.message}`);
  } else {
    throw caught;
  }
}
};
```

- Einzelheiten zur API finden Sie [UnmonitorInstances](#) in der AWS SDK für JavaScript API-Referenz.

Szenarien

Erstellen und Verwalten eines ausfallsicheren Services

Das folgende Codebeispiel zeigt, wie Sie einen Webservice mit Load Balancing erstellen, der Buch-, Film- und Liedempfehlungen zurückgibt. Das Beispiel zeigt, wie der Service auf Fehler reagiert und wie der Service für mehr Ausfallsicherheit umstrukturiert werden kann.

- Verwenden Sie eine Amazon EC2 Auto Scaling Scaling-Gruppe, um Amazon Elastic Compute Cloud (Amazon EC2) -Instances auf der Grundlage einer Startvorlage zu erstellen und die Anzahl der Instances in einem bestimmten Bereich zu halten.
- Verarbeiten und verteilen Sie HTTP-Anfragen mit Elastic Load Balancing.
- Überwachen Sie den Zustand von Instances in einer Auto-Scaling-Gruppe und leiten Sie Anfragen nur an fehlerfreie Instances weiter.

- Führen Sie auf jeder EC2 Instanz einen Python-Webserver aus, um HTTP-Anfragen zu verarbeiten. Der Webserver reagiert mit Empfehlungen und Zustandsprüfungen.
- Simulieren Sie einen Empfehlungsservice mit einer Amazon DynamoDB-Tabelle.
- Steuern Sie die Antwort des Webserver auf Anfragen und Zustandsprüfungen, indem Sie die AWS Systems Manager Parameter aktualisieren.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Führen Sie ein interaktives Szenario an einer Eingabeaufforderung aus.

```
#!/usr/bin/env node
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import {
  Scenario,
  parseScenarioArgs,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * The workflow steps are split into three stages:
 *   - deploy
 *   - demo
 *   - destroy
 *
 * Each of these stages has a corresponding file prefixed with steps-*.
 */
import { deploySteps } from "./steps-deploy.js";
import { demoSteps } from "./steps-demo.js";
import { destroySteps } from "./steps-destroy.js";

/**
 * The context is passed to every scenario. Scenario steps
 * will modify the context.
 */
```

```

const context = {};

/**
 * Three Scenarios are created for the workflow. A Scenario is an orchestration
 * class
 * that simplifies running a series of steps.
 */
export const scenarios = {
  // Deploys all resources necessary for the workflow.
  deploy: new Scenario("Resilient Workflow - Deploy", deploySteps, context),
  // Demonstrates how a fragile web service can be made more resilient.
  demo: new Scenario("Resilient Workflow - Demo", demoSteps, context),
  // Destroys the resources created for the workflow.
  destroy: new Scenario("Resilient Workflow - Destroy", destroySteps, context),
};

// Call function if run directly
import { fileURLToPath } from "node:url";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios, {
    name: "Resilient Workflow",
    synopsis:
      "node index.js --scenario <deploy | demo | destroy> [-h|--help] [-y|--yes] [-v|--verbose]",
    description: "Deploy and interact with scalable EC2 instances.",
  });
}

```

Erstellen Sie Schritte, um alle Ressourcen bereitzustellen.

```

import { join } from "node:path";
import { readFileSync, writeFileSync } from "node:fs";
import axios from "axios";

import {
  BatchWriteItemCommand,
  CreateTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {

```

```
    EC2Client,
    CreateKeyPairCommand,
    CreateLaunchTemplateCommand,
    DescribeAvailabilityZonesCommand,
    DescribeVpcsCommand,
    DescribeSubnetsCommand,
    DescribeSecurityGroupsCommand,
    AuthorizeSecurityGroupIngressCommand,
} from "@aws-sdk/client-ec2";
import {
    IAMClient,
    CreatePolicyCommand,
    CreateRoleCommand,
    CreateInstanceProfileCommand,
    AddRoleToInstanceProfileCommand,
    AttachRolePolicyCommand,
    waitUntilInstanceProfileExists,
} from "@aws-sdk/client-iam";
import { SSMClient, GetParameterCommand } from "@aws-sdk/client-ssm";
import {
    CreateAutoScalingGroupCommand,
    AutoScalingClient,
    AttachLoadBalancerTargetGroupsCommand,
} from "@aws-sdk/client-auto-scaling";
import {
    CreateListenerCommand,
    CreateLoadBalancerCommand,
    CreateTargetGroupCommand,
    ElasticLoadBalancingV2Client,
    waitUntilLoadBalancerAvailable,
} from "@aws-sdk/client-elastic-load-balancing-v2";

import {
    ScenarioOutput,
    ScenarioInput,
    ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { saveState } from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES, RESOURCES_PATH, ROOT } from "../constants.js";
import { initParamsSteps } from "../steps-reset-params.js";

/**
```

```
* @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
*/
export const deploySteps = [
  new ScenarioOutput("introduction", MESSAGES.introduction, { header: true }),
  new ScenarioInput("confirmDeployment", MESSAGES.confirmDeployment, {
    type: "confirm",
  }),
  new ScenarioAction(
    "handleConfirmDeployment",
    (c) => c.confirmDeployment === false && process.exit(),
  ),
  new ScenarioOutput(
    "creatingTable",
    MESSAGES.creatingTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioAction("createTable", async () => {
    const client = new DynamoDBClient({});
    await client.send(
      new CreateTableCommand({
        TableName: NAMES.tableName,
        ProvisionedThroughput: {
          ReadCapacityUnits: 5,
          WriteCapacityUnits: 5,
        },
        AttributeDefinitions: [
          {
            AttributeName: "MediaType",
            AttributeType: "S",
          },
          {
            AttributeName: "ItemId",
            AttributeType: "N",
          },
        ],
        KeySchema: [
          {
            AttributeName: "MediaType",
            KeyType: "HASH",
          },
          {
            AttributeName: "ItemId",
            KeyType: "RANGE",
          },
        ],
      })
    );
  })
];
```

```

    }},
  );
  await waitUntilTableExists({ client }, { TableName: NAMES.tableName });
}),
new ScenarioOutput(
  "createdTable",
  MESSAGES.createdTable.replace("${TABLE_NAME}", NAMES.tableName),
),
new ScenarioOutput(
  "populatingTable",
  MESSAGES.populatingTable.replace("${TABLE_NAME}", NAMES.tableName),
),
new ScenarioAction("populateTable", () => {
  const client = new DynamoDBClient({});
  /**
   * @type {{ default: import("@aws-sdk/client-dynamodb").PutRequest['Item'][] }}
   */
  const recommendations = JSON.parse(
    readFileSync(join(RESOURCES_PATH, "recommendations.json")),
  );

  return client.send(
    new BatchWriteItemCommand({
      RequestItems: {
        [NAMES.tableName]: recommendations.map((item) => ({
          PutRequest: { Item: item },
        })),
      },
    }),
  );
}),
new ScenarioOutput(
  "populatedTable",
  MESSAGES.populatedTable.replace("${TABLE_NAME}", NAMES.tableName),
),
new ScenarioOutput(
  "creatingKeyPair",
  MESSAGES.creatingKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
),
new ScenarioAction("createKeyPair", async () => {
  const client = new EC2Client({});
  const { KeyMaterial } = await client.send(
    new CreateKeyPairCommand({
      KeyName: NAMES.keyPairName,

```

```

    }},
  );

  writeFileSync(`${NAMES.keyPairName}.pem`, KeyMaterial, { mode: 0o600 });
}),
new ScenarioOutput(
  "createdKeyPair",
  MESSAGES.createdKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
),
new ScenarioOutput(
  "creatingInstancePolicy",
  MESSAGES.creatingInstancePolicy.replace(
    "${INSTANCE_POLICY_NAME}",
    NAMES.instancePolicyName,
  ),
),
new ScenarioAction("createInstancePolicy", async (state) => {
  const client = new IAMClient({});
  const {
    Policy: { Arn },
  } = await client.send(
    new CreatePolicyCommand({
      PolicyName: NAMES.instancePolicyName,
      PolicyDocument: readFileSync(
        join(RESOURCES_PATH, "instance_policy.json"),
      ),
    }),
  ),
  state.instancePolicyArn = Arn;
}),
new ScenarioOutput("createdInstancePolicy", (state) =>
  MESSAGES.createdInstancePolicy
    .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
    .replace("${INSTANCE_POLICY_ARN}", state.instancePolicyArn),
),
new ScenarioOutput(
  "creatingInstanceRole",
  MESSAGES.creatingInstanceRole.replace(
    "${INSTANCE_ROLE_NAME}",
    NAMES.instanceRoleName,
  ),
),
new ScenarioAction("createInstanceRole", () => {
  const client = new IAMClient({});

```



```

    return client.send(
      new CreateRoleCommand({
        RoleName: NAMES.instanceRoleName,
        AssumeRolePolicyDocument: readFileSync(
          join(ROOT, "assume-role-policy.json"),
        ),
      }),
    );
  }},
  new ScenarioOutput(
    "createdInstanceRole",
    MESSAGES.createdInstanceRole.replace(
      "${INSTANCE_ROLE_NAME}",
      NAMES.instanceRoleName,
    ),
  ),
  new ScenarioOutput(
    "attachingPolicyToRole",
    MESSAGES.attachingPolicyToRole
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName)
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName),
  ),
  new ScenarioAction("attachPolicyToRole", async (state) => {
    const client = new IAMClient({});
    await client.send(
      new AttachRolePolicyCommand({
        RoleName: NAMES.instanceRoleName,
        PolicyArn: state.instancePolicyArn,
      }),
    );
  }},
  new ScenarioOutput(
    "attachedPolicyToRole",
    MESSAGES.attachedPolicyToRole
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
  ),
  new ScenarioOutput(
    "creatingInstanceProfile",
    MESSAGES.creatingInstanceProfile.replace(
      "${INSTANCE_PROFILE_NAME}",
      NAMES.instanceProfileName,
    ),
  ),
),

```

```
new ScenarioAction("createInstanceProfile", async (state) => {
  const client = new IAMClient({});
  const {
    InstanceProfile: { Arn },
  } = await client.send(
    new CreateInstanceProfileCommand({
      InstanceProfileName: NAMES.instanceProfileName,
    }),
  );
  state.instanceProfileArn = Arn;

  await waitUntilInstanceProfileExists(
    { client },
    { InstanceProfileName: NAMES.instanceProfileName },
  );
}),
new ScenarioOutput("createdInstanceProfile", (state) =>
  MESSAGES.createdInstanceProfile
    .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
    .replace("${INSTANCE_PROFILE_ARN}", state.instanceProfileArn),
),
new ScenarioOutput(
  "addingRoleToInstanceProfile",
  MESSAGES.addingRoleToInstanceProfile
    .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
    .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
),
new ScenarioAction("addRoleToInstanceProfile", () => {
  const client = new IAMClient({});
  return client.send(
    new AddRoleToInstanceProfileCommand({
      RoleName: NAMES.instanceRoleName,
      InstanceProfileName: NAMES.instanceProfileName,
    }),
  );
}),
new ScenarioOutput(
  "addedRoleToInstanceProfile",
  MESSAGES.addedRoleToInstanceProfile
    .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
    .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
),
...initParamsSteps,
new ScenarioOutput("creatingLaunchTemplate", MESSAGES.creatingLaunchTemplate),
```

```

new ScenarioAction("createLaunchTemplate", async () => {
  const ssmClient = new SSMClient({});
  const { Parameter } = await ssmClient.send(
    new GetParameterCommand({
      Name: "/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2",
    }),
  );
  const ec2Client = new EC2Client({});
  await ec2Client.send(
    new CreateLaunchTemplateCommand({
      LaunchTemplateName: NAMES.launchTemplateName,
      LaunchTemplateData: {
        InstanceType: "t3.micro",
        ImageId: Parameter.Value,
        IamInstanceProfile: { Name: NAMES.instanceProfileName },
        UserData: readFileSync(
          join(RESOURCES_PATH, "server_startup_script.sh"),
        ).toString("base64"),
        KeyName: NAMES.keyPairName,
      },
    }),
  );
},
new ScenarioOutput(
  "createdLaunchTemplate",
  MESSAGES.createdLaunchTemplate.replace(
    "${LAUNCH_TEMPLATE_NAME}",
    NAMES.launchTemplateName,
  ),
),
new ScenarioOutput(
  "creatingAutoScalingGroup",
  MESSAGES.creatingAutoScalingGroup.replace(
    "${AUTO_SCALING_GROUP_NAME}",
    NAMES.autoScalingGroupName,
  ),
),
new ScenarioAction("createAutoScalingGroup", async (state) => {
  const ec2Client = new EC2Client({});
  const { AvailabilityZones } = await ec2Client.send(
    new DescribeAvailabilityZonesCommand({}),
  );
  state.availabilityZoneNames = AvailabilityZones.map((az) => az.ZoneName);
  const autoScalingClient = new AutoScalingClient({});

```

```

    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      autoScalingClient.send(
        new CreateAutoScalingGroupCommand({
          AvailabilityZones: state.availabilityZoneNames,
          AutoScalingGroupName: NAMES.autoScalingGroupName,
          LaunchTemplate: {
            LaunchTemplateName: NAMES.launchTemplateName,
            Version: "$Default",
          },
          MinSize: 3,
          MaxSize: 3,
        }),
      ),
    );
  }},
  new ScenarioOutput(
    "createdAutoScalingGroup",
    /**
     * @param {{ availabilityZoneNames: string[] }} state
     */
    (state) =>
      MESSAGES.createdAutoScalingGroup
        .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName)
        .replace(
          "${AVAILABILITY_ZONE_NAMES}",
          state.availabilityZoneNames.join(", "),
        ),
  ),
  new ScenarioInput("confirmContinue", MESSAGES.confirmContinue, {
    type: "confirm",
  }),
  new ScenarioOutput("loadBalancer", MESSAGES.loadBalancer),
  new ScenarioOutput("gettingVpc", MESSAGES.gettingVpc),
  new ScenarioAction("getVpc", async (state) => {
    const client = new EC2Client({});
    const { Vpcs } = await client.send(
      new DescribeVpcsCommand({
        Filters: [{ Name: "is-default", Values: ["true"] }],
      }),
    );
    state.defaultVpc = Vpcs[0].VpcId;
  }),
  new ScenarioOutput("gotVpc", (state) =>
    MESSAGES.gotVpc.replace("${VPC_ID}", state.defaultVpc),
  ),

```

```

    ),
    new ScenarioOutput("gettingSubnets", MESSAGES.gettingSubnets),
    new ScenarioAction("getSubnets", async (state) => {
      const client = new EC2Client({});
      const { Subnets } = await client.send(
        new DescribeSubnetsCommand({
          Filters: [
            { Name: "vpc-id", Values: [state.defaultVpc] },
            { Name: "availability-zone", Values: state.availabilityZoneNames },
            { Name: "default-for-az", Values: ["true"] },
          ],
        })
      );
      state.subnets = Subnets.map((subnet) => subnet.SubnetId);
    }),
    new ScenarioOutput(
      "gotSubnets",
      /**
       * @param {{ subnets: string[] }} state
       */
      (state) =>
        MESSAGES.gotSubnets.replace("${SUBNETS}", state.subnets.join(", ")),
    ),
    new ScenarioOutput(
      "creatingLoadBalancerTargetGroup",
      MESSAGES.creatingLoadBalancerTargetGroup.replace(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      ),
    ),
    new ScenarioAction("createLoadBalancerTargetGroup", async (state) => {
      const client = new ElasticLoadBalancingV2Client({});
      const { TargetGroups } = await client.send(
        new CreateTargetGroupCommand({
          Name: NAMES.loadBalancerTargetGroupName,
          Protocol: "HTTP",
          Port: 80,
          HealthCheckPath: "/healthcheck",
          HealthCheckIntervalSeconds: 10,
          HealthCheckTimeoutSeconds: 5,
          HealthyThresholdCount: 2,
          UnhealthyThresholdCount: 2,
          VpcId: state.defaultVpc,
        })
      ),
    ),

```

```

    );
    const targetGroup = TargetGroups[0];
    state.targetGroupArn = targetGroup.TargetGroupArn;
    state.targetGroupProtocol = targetGroup.Protocol;
    state.targetGroupPort = targetGroup.Port;
  }},
  new ScenarioOutput(
    "createdLoadBalancerTargetGroup",
    MESSAGES.createdLoadBalancerTargetGroup.replace(
      "${TARGET_GROUP_NAME}",
      NAMES.loadBalancerTargetGroupName,
    ),
  ),
  new ScenarioOutput(
    "creatingLoadBalancer",
    MESSAGES.creatingLoadBalancer.replace("${LB_NAME}", NAMES.loadBalancerName),
  ),
  new ScenarioAction("createLoadBalancer", async (state) => {
    const client = new ElasticLoadBalancingV2Client({});
    const { LoadBalancers } = await client.send(
      new CreateLoadBalancerCommand({
        Name: NAMES.loadBalancerName,
        Subnets: state.subnets,
      })
    );
    state.loadBalancerDns = LoadBalancers[0].DNSName;
    state.loadBalancerArn = LoadBalancers[0].LoadBalancerArn;
    await waitUntilLoadBalancerAvailable(
      { client },
      { Names: [NAMES.loadBalancerName] },
    );
  }),
  new ScenarioOutput("createdLoadBalancer", (state) =>
    MESSAGES.createdLoadBalancer
      .replace("${LB_NAME}", NAMES.loadBalancerName)
      .replace("${DNS_NAME}", state.loadBalancerDns),
  ),
  new ScenarioOutput(
    "creatingListener",
    MESSAGES.creatingLoadBalancerListener
      .replace("${LB_NAME}", NAMES.loadBalancerName)
      .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName),
  ),
  new ScenarioAction("createListener", async (state) => {

```

```

const client = new ElasticLoadBalancingV2Client({});
const { Listeners } = await client.send(
  new CreateListenerCommand({
    LoadBalancerArn: state.loadBalancerArn,
    Protocol: state.targetGroupProtocol,
    Port: state.targetGroupPort,
    DefaultActions: [
      { Type: "forward", TargetGroupArn: state.targetGroupArn },
    ],
  }),
);
const listener = Listeners[0];
state.loadBalancerListenerArn = listener.ListenerArn;
}),
new ScenarioOutput("createdListener", (state) =>
  MESSAGES.createdLoadBalancerListener.replace(
    "${LB_LISTENER_ARN}",
    state.loadBalancerListenerArn,
  ),
),
new ScenarioOutput(
  "attachingLoadBalancerTargetGroup",
  MESSAGES.attachingLoadBalancerTargetGroup
    .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName)
    .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName),
),
new ScenarioAction("attachLoadBalancerTargetGroup", async (state) => {
  const client = new AutoScalingClient({});
  await client.send(
    new AttachLoadBalancerTargetGroupsCommand({
      AutoScalingGroupName: NAMES.autoScalingGroupName,
      TargetGroupARNs: [state.targetGroupArn],
    }),
  );
}),
new ScenarioOutput(
  "attachedLoadBalancerTargetGroup",
  MESSAGES.attachedLoadBalancerTargetGroup,
),
new ScenarioOutput("verifyingInboundPort", MESSAGES.verifyingInboundPort),
new ScenarioAction(
  "verifyInboundPort",
  /**
   *

```

```

    * @param {{ defaultSecurityGroup: import('@aws-sdk/client-ec2').SecurityGroup}}
state
    */
    async (state) => {
        const client = new EC2Client({});
        const { SecurityGroups } = await client.send(
            new DescribeSecurityGroupsCommand({
                Filters: [{ Name: "group-name", Values: ["default"] }],
            }),
        );
        if (!SecurityGroups) {
            state.verifyInboundPortError = new Error(MESSAGES.noSecurityGroups);
        }
        state.defaultSecurityGroup = SecurityGroups[0];

        /**
         * @type {string}
         */
        const ipResponse = (await axios.get("http://checkip.amazonaws.com")).data;
        state.myIp = ipResponse.trim();
        const myIpRules = state.defaultSecurityGroup.IpPermissions.filter(
            ({ IpRanges }) =>
                IpRanges.some(
                    ({ CidrIp }) =>
                        CidrIp.startsWith(state.myIp) || CidrIp === "0.0.0.0/0",
                ),
        )
            .filter(({ IpProtocol }) => IpProtocol === "tcp")
            .filter(({ FromPort }) => FromPort === 80);

        state.myIpRules = myIpRules;
    },
),
new ScenarioOutput(
    "verifiedInboundPort",
    /**
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
        if (state.myIpRules.length > 0) {
            return MESSAGES.foundIpRules.replace(
                "${IP_RULES}",
                JSON.stringify(state.myIpRules, null, 2),
            );
        }
    }
);

```



```

    }
    return MESSAGES.noIpRules;
  },
),
new ScenarioInput(
  "shouldAddInboundRule",
  /**
   * @param {{ myIpRules: any[] }} state
   */
  (state) => {
    if (state.myIpRules.length > 0) {
      return false;
    }
    return MESSAGES.noIpRules;
  },
  { type: "confirm" },
),
new ScenarioAction(
  "addInboundRule",
  /**
   * @param {{ defaultSecurityGroup: import('@aws-sdk/client-ec2').SecurityGroup }} state
   */
  async (state) => {
    if (!state.shouldAddInboundRule) {
      return;
    }

    const client = new EC2Client({});
    await client.send(
      new AuthorizeSecurityGroupIngressCommand({
        GroupId: state.defaultSecurityGroup.GroupId,
        CidrIp: `${state.myIp}/32`,
        FromPort: 80,
        ToPort: 80,
        IpProtocol: "tcp",
      }),
    );
  },
),
new ScenarioOutput("addedInboundRule", (state) => {
  if (state.shouldAddInboundRule) {
    return MESSAGES.addedInboundRule.replace("${IP_ADDRESS}", state.myIp);
  }
})

```

```

    return false;
  })),
  new ScenarioOutput("verifyingEndpoint", (state) =>
    MESSAGES.verifyingEndpoint.replace("${DNS_NAME}", state.loadBalancerDns),
  ),
  new ScenarioAction("verifyEndpoint", async (state) => {
    try {
      const response = await retry({ intervalInMs: 2000, maxRetries: 30 }, () =>
        axios.get(`http://${state.loadBalancerDns}`),
      );
      state.endpointResponse = JSON.stringify(response.data, null, 2);
    } catch (e) {
      state.verifyEndpointError = e;
    }
  })),
  new ScenarioOutput("verifiedEndpoint", (state) => {
    if (state.verifyEndpointError) {
      console.error(state.verifyEndpointError);
    } else {
      return MESSAGES.verifiedEndpoint.replace(
        "${ENDPOINT_RESPONSE}",
        state.endpointResponse,
      );
    }
  })),
  saveState,
];

```

Erstellen Sie Schritte, um die Demo auszuführen.

```

import { readFileSync } from "node:fs";
import { join } from "node:path";

import axios from "axios";

import {
  DescribeTargetGroupsCommand,
  DescribeTargetHealthCommand,
  ElasticLoadBalancingV2Client,
} from "@aws-sdk/client-elastic-load-balancing-v2";
import {
  DescribeInstanceInformationCommand,

```

```
PutParameterCommand,
SSMClient,
SendCommandCommand,
} from "@aws-sdk/client-ssm";
import {
  IAMClient,
  CreatePolicyCommand,
  CreateRoleCommand,
  AttachRolePolicyCommand,
  CreateInstanceProfileCommand,
  AddRoleToInstanceProfileCommand,
  waitUntilInstanceProfileExists,
} from "@aws-sdk/client-iam";
import {
  AutoScalingClient,
  DescribeAutoScalingGroupsCommand,
  TerminateInstanceInAutoScalingGroupCommand,
} from "@aws-sdk/client-auto-scaling";
import {
  DescribeIamInstanceProfileAssociationsCommand,
  EC2Client,
  RebootInstancesCommand,
  ReplaceIamInstanceProfileAssociationCommand,
} from "@aws-sdk/client-ec2";

import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/scenario.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES, RESOURCES_PATH } from "../constants.js";
import { findLoadBalancer } from "../shared.js";

const getRecommendation = new ScenarioAction(
  "getRecommendation",
  async (state) => {
    const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
    if (loadBalancer) {
      state.loadBalancerDnsName = loadBalancer.DNSName;
      try {
        state.recommendation = (
          await axios.get(`http://${state.loadBalancerDnsName}`)
        ).data;
      } catch (error) {
        console.error(error);
      }
    }
  }
);
```

```

        ).data;
    } catch (e) {
        state.recommendation = e instanceof Error ? e.message : e;
    }
    } else {
        throw new Error(MESSAGES.demoFindLoadBalancerError);
    }
    },
    );

const getRecommendationResult = new ScenarioOutput(
    "getRecommendationResult",
    (state) =>
        `Recommendation:\n${JSON.stringify(state.recommendation, null, 2)}`,
    { preformatted: true },
    );

const getHealthCheck = new ScenarioAction("getHealthCheck", async (state) => {
    const client = new ElasticLoadBalancingV2Client({});
    const { TargetGroups } = await client.send(
        new DescribeTargetGroupsCommand({
            Names: [NAMES.loadBalancerTargetGroupName],
        }),
    );
    );

    const { TargetHealthDescriptions } = await client.send(
        new DescribeTargetHealthCommand({
            TargetGroupArn: TargetGroups[0].TargetGroupArn,
        }),
    );
    state.targetHealthDescriptions = TargetHealthDescriptions;
});

const getHealthCheckResult = new ScenarioOutput(
    "getHealthCheckResult",
    /**
     * @param {{ targetHealthDescriptions: import('@aws-sdk/client-elastic-load-
    balancing-v2').TargetHealthDescription[] }} state
     */
    (state) => {
        const status = state.targetHealthDescriptions
            .map((th) => `${th.Target.Id}: ${th.TargetHealth.State}`)
            .join("\n");
        return `Health check:\n${status}`;
    }
);

```

```
    },
    { preformatted: true },
  );

const loadBalancerLoop = new ScenarioAction(
  "loadBalancerLoop",
  getRecommendation.action,
  {
    whileConfig: {
      whileFn: ({ loadBalancerCheck }) => loadBalancerCheck,
      input: new ScenarioInput(
        "loadBalancerCheck",
        MESSAGES.demoLoadBalancerCheck,
        {
          type: "confirm",
        },
      ),
      output: getRecommendationResult,
    },
  },
);

const healthCheckLoop = new ScenarioAction(
  "healthCheckLoop",
  getHealthCheck.action,
  {
    whileConfig: {
      whileFn: ({ healthCheck }) => healthCheck,
      input: new ScenarioInput("healthCheck", MESSAGES.demoHealthCheck, {
        type: "confirm",
      }),
      output: getHealthCheckResult,
    },
  },
);

const statusSteps = [
  getRecommendation,
  getRecommendationResult,
  getHealthCheck,
  getHealthCheckResult,
];

/**
```

```

* @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
*/
export const demoSteps = [
  new ScenarioOutput("header", MESSAGES.demoHeader, { header: true }),
  new ScenarioOutput("sanityCheck", MESSAGES.demoSanityCheck),
  ...statusSteps,
  new ScenarioInput(
    "brokenDependencyConfirmation",
    MESSAGES.demoBrokenDependencyConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("brokenDependency", async (state) => {
    if (!state.brokenDependencyConfirmation) {
      process.exit();
    } else {
      const client = new SSMClient({});
      state.badTableName = `fake-table-${Date.now()}`;
      await client.send(
        new PutParameterCommand({
          Name: NAMES.ssmTableNameKey,
          Value: state.badTableName,
          Overwrite: true,
          Type: "String",
        }),
      );
    }
  }),
  new ScenarioOutput("testBrokenDependency", (state) =>
    MESSAGES.demoTestBrokenDependency.replace(
      "${TABLE_NAME}",
      state.badTableName,
    ),
  ),
  ...statusSteps,
  new ScenarioInput(
    "staticResponseConfirmation",
    MESSAGES.demoStaticResponseConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("staticResponse", async (state) => {
    if (!state.staticResponseConfirmation) {
      process.exit();
    } else {
      const client = new SSMClient({});

```

```

    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmFailureResponseKey,
        Value: "static",
        Overwrite: true,
        Type: "String",
      }),
    );
  }
}),
new ScenarioOutput("testStaticResponse", MESSAGES.demoTestStaticResponse),
...statusSteps,
new ScenarioInput(
  "badCredentialsConfirmation",
  MESSAGES.demoBadCredentialsConfirmation,
  { type: "confirm" },
),
new ScenarioAction("badCredentialsExit", (state) => {
  if (!state.badCredentialsConfirmation) {
    process.exit();
  }
}),
new ScenarioAction("fixDynamoDBName", async () => {
  const client = new SSMClient({});
  await client.send(
    new PutParameterCommand({
      Name: NAMES.ssmTableNameKey,
      Value: NAMES.tableName,
      Overwrite: true,
      Type: "String",
    }),
  );
}),
new ScenarioAction(
  "badCredentials",
  /**
   * @param {{ targetInstance: import('@aws-sdk/client-auto-scaling').Instance }}
state
  */
  async (state) => {
    await createSsmOnlyInstanceProfile();
    const autoScalingClient = new AutoScalingClient({});
    const { AutoScalingGroups } = await autoScalingClient.send(
      new DescribeAutoScalingGroupsCommand({

```

```
    AutoScalingGroupNames: [NAMES.autoScalingGroupName],
  })),
);
state.targetInstance = AutoScalingGroups[0].Instances[0];
const ec2Client = new EC2Client({});
const { IamInstanceProfileAssociations } = await ec2Client.send(
  new DescribeIamInstanceProfileAssociationsCommand({
    Filters: [
      { Name: "instance-id", Values: [state.targetInstance.InstanceId] },
    ],
  })),
);
state.instanceProfileAssociationId =
  IamInstanceProfileAssociations[0].AssociationId;
await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
  ec2Client.send(
    new ReplaceIamInstanceProfileAssociationCommand({
      AssociationId: state.instanceProfileAssociationId,
      IamInstanceProfile: { Name: NAMES.ssmOnlyInstanceProfileName },
    }),
  ),
);

await ec2Client.send(
  new RebootInstancesCommand({
    InstanceIds: [state.targetInstance.InstanceId],
  })),
);

const ssmClient = new SSMClient({});
await retry({ intervalInMs: 20000, maxRetries: 15 }, async () => {
  const { InstanceInformationList } = await ssmClient.send(
    new DescribeInstanceInformationCommand({}),
  );

  const instance = InstanceInformationList.find(
    (info) => info.InstanceId === state.targetInstance.InstanceId,
  );

  if (!instance) {
    throw new Error("Instance not found.");
  }
});
```



```

    await ssmClient.send(
      new SendCommandCommand({
        InstanceIds: [state.targetInstance.InstanceId],
        DocumentName: "AWS-RunShellScript",
        Parameters: { commands: ["cd / && sudo python3 server.py 80"] },
      }),
    );
  },
),
new ScenarioOutput(
  "testBadCredentials",
  /**
   * @param {{ targetInstance: import('@aws-sdk/client-ssm').InstanceInformation}}
state
  */
  (state) =>
    MESSAGES.demoTestBadCredentials.replace(
      "${INSTANCE_ID}",
      state.targetInstance.InstanceId,
    ),
),
loadBalancerLoop,
new ScenarioInput(
  "deepHealthCheckConfirmation",
  MESSAGES.demoDeepHealthCheckConfirmation,
  { type: "confirm" },
),
new ScenarioAction("deepHealthCheckExit", (state) => {
  if (!state.deepHealthCheckConfirmation) {
    process.exit();
  }
})),
new ScenarioAction("deepHealthCheck", async () => {
  const client = new SSMClient({});
  await client.send(
    new PutParameterCommand({
      Name: NAMES.ssmHealthCheckKey,
      Value: "deep",
      Overwrite: true,
      Type: "String",
    }),
  );
})),
new ScenarioOutput("testDeepHealthCheck", MESSAGES.demoTestDeepHealthCheck),

```

```

healthCheckLoop,
loadBalancerLoop,
new ScenarioInput(
  "killInstanceConfirmation",
  /**
   * @param {{ targetInstance: import('@aws-sdk/client-
ssm').InstanceInformation }} state
   */
  (state) =>
    MESSAGES.demoKillInstanceConfirmation.replace(
      "${INSTANCE_ID}",
      state.targetInstance.InstanceId,
    ),
  { type: "confirm" },
),
new ScenarioAction("killInstanceExit", (state) => {
  if (!state.killInstanceConfirmation) {
    process.exit();
  }
}),
new ScenarioAction(
  "killInstance",
  /**
   * @param {{ targetInstance: import('@aws-sdk/client-
ssm').InstanceInformation }} state
   */
  async (state) => {
    const client = new AutoScalingClient({});
    await client.send(
      new TerminateInstanceInAutoScalingGroupCommand({
        InstanceId: state.targetInstance.InstanceId,
        ShouldDecrementDesiredCapacity: false,
      }),
    );
  },
),
new ScenarioOutput("testKillInstance", MESSAGES.demoTestKillInstance),
healthCheckLoop,
loadBalancerLoop,
new ScenarioInput("failOpenConfirmation", MESSAGES.demoFailOpenConfirmation, {
  type: "confirm",
}),
new ScenarioAction("failOpenExit", (state) => {
  if (!state.failOpenConfirmation) {

```

```

        process.exit();
    }
}),
new ScenarioAction("failOpen", () => {
    const client = new SSMClient({});
    return client.send(
        new PutParameterCommand({
            Name: NAMES.ssmTableNameKey,
            Value: `fake-table-${Date.now()}`,
            Overwrite: true,
            Type: "String",
        }),
    );
}),
new ScenarioOutput("testFailOpen", MESSAGES.demoFailOpenTest),
healthCheckLoop,
loadBalancerLoop,
new ScenarioInput(
    "resetTableConfirmation",
    MESSAGES.demoResetTableConfirmation,
    { type: "confirm" },
),
new ScenarioAction("resetTableExit", (state) => {
    if (!state.resetTableConfirmation) {
        process.exit();
    }
}),
new ScenarioAction("resetTable", async () => {
    const client = new SSMClient({});
    await client.send(
        new PutParameterCommand({
            Name: NAMES.ssmTableNameKey,
            Value: NAMES.tableName,
            Overwrite: true,
            Type: "String",
        }),
    );
}),
new ScenarioOutput("testResetTable", MESSAGES.demoTestResetTable),
healthCheckLoop,
loadBalancerLoop,
];

async function createSsmOnlyInstanceProfile() {

```

```
const iamClient = new IAMClient({});
const { Policy } = await iamClient.send(
  new CreatePolicyCommand({
    PolicyName: NAMES.ssmOnlyPolicyName,
    PolicyDocument: readFileSync(
      join(RESOURCES_PATH, "ssm_only_policy.json"),
    ),
  }),
);
await iamClient.send(
  new CreateRoleCommand({
    RoleName: NAMES.ssmOnlyRoleName,
    AssumeRolePolicyDocument: JSON.stringify({
      Version: "2012-10-17",
      Statement: [
        {
          Effect: "Allow",
          Principal: { Service: "ec2.amazonaws.com" },
          Action: "sts:AssumeRole",
        },
      ],
    }),
  }),
);
await iamClient.send(
  new AttachRolePolicyCommand({
    RoleName: NAMES.ssmOnlyRoleName,
    PolicyArn: Policy.Arn,
  }),
);
await iamClient.send(
  new AttachRolePolicyCommand({
    RoleName: NAMES.ssmOnlyRoleName,
    PolicyArn: "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore",
  }),
);
const { InstanceProfile } = await iamClient.send(
  new CreateInstanceProfileCommand({
    InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
  }),
);
await waitUntilInstanceProfileExists(
  { client: iamClient },
  { InstanceProfileName: NAMES.ssmOnlyInstanceProfileName },
);
```

```

);
await iamClient.send(
  new AddRoleToInstanceProfileCommand({
    InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
    RoleName: NAMES.ssmOnlyRoleName,
  }),
);

return InstanceProfile;
}

```

Erstellen Sie Schritte, um alle Ressourcen zu vernichten.

```

import { unlinkSync } from "node:fs";

import { DynamoDBClient, DeleteTableCommand } from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  DeleteKeyPairCommand,
  DeleteLaunchTemplateCommand,
  RevokeSecurityGroupIngressCommand,
} from "@aws-sdk/client-ec2";
import {
  IAMClient,
  DeleteInstanceProfileCommand,
  RemoveRoleFromInstanceProfileCommand,
  DeletePolicyCommand,
  DeleteRoleCommand,
  DetachRolePolicyCommand,
  paginateListPolicies,
} from "@aws-sdk/client-iam";
import {
  AutoScalingClient,
  DeleteAutoScalingGroupCommand,
  TerminateInstanceInAutoScalingGroupCommand,
  UpdateAutoScalingGroupCommand,
  paginateDescribeAutoScalingGroups,
} from "@aws-sdk/client-auto-scaling";
import {
  DeleteLoadBalancerCommand,
  DeleteTargetGroupCommand,
  DescribeTargetGroupsCommand,

```

```

    ElasticLoadBalancingV2Client,
  } from "@aws-sdk/client-elastic-load-balancing-v2";

import {
  ScenarioOutput,
  ScenarioInput,
  ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { loadState } from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES } from "../constants.js";
import { findLoadBalancer } from "../shared.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const destroySteps = [
  loadState,
  new ScenarioInput("destroy", MESSAGES.destroy, { type: "confirm" }),
  new ScenarioAction(
    "abort",
    (state) => state.destroy === false && process.exit(),
  ),
  new ScenarioAction("deleteTable", async (c) => {
    try {
      const client = new DynamoDBClient({});
      await client.send(new DeleteTableCommand({ TableName: NAMES.tableName }));
    } catch (e) {
      c.deleteTableError = e;
    }
  }),
  new ScenarioOutput("deleteTableResult", (state) => {
    if (state.deleteTableError) {
      console.error(state.deleteTableError);
      return MESSAGES.deleteTableError.replace(
        "${TABLE_NAME}",
        NAMES.tableName,
      );
    }
    return MESSAGES.deletedTable.replace("${TABLE_NAME}", NAMES.tableName);
  }),
  new ScenarioAction("deleteKeyPair", async (state) => {
    try {

```

```

    const client = new EC2Client({});
    await client.send(
      new DeleteKeyPairCommand({ KeyName: NAMES.keyPairName }),
    );
    unlinkSync(`${NAMES.keyPairName}.pem`);
  } catch (e) {
    state.deleteKeyPairError = e;
  }
}),
new ScenarioOutput("deleteKeyPairResult", (state) => {
  if (state.deleteKeyPairError) {
    console.error(state.deleteKeyPairError);
    return MESSAGES.deleteKeyPairError.replace(
      "${KEY_PAIR_NAME}",
      NAMES.keyPairName,
    );
  }
  return MESSAGES.deletedKeyPair.replace(
    "${KEY_PAIR_NAME}",
    NAMES.keyPairName,
  );
}),
new ScenarioAction("detachPolicyFromRole", async (state) => {
  try {
    const client = new IAMClient({});
    const policy = await findPolicy(NAMES.instancePolicyName);

    if (!policy) {
      state.detachPolicyFromRoleError = new Error(
        `Policy ${NAMES.instancePolicyName} not found.`,
      );
    } else {
      await client.send(
        new DetachRolePolicyCommand({
          RoleName: NAMES.instanceRoleName,
          PolicyArn: policy.Arn,
        }),
      );
    }
  } catch (e) {
    state.detachPolicyFromRoleError = e;
  }
}),
new ScenarioOutput("detachedPolicyFromRole", (state) => {

```

```
    if (state.detachPolicyFromRoleError) {
      console.error(state.detachPolicyFromRoleError);
      return MESSAGES.detachPolicyFromRoleError
        .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    }
    return MESSAGES.detachedPolicyFromRole
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
  })),
  new ScenarioAction("deleteInstancePolicy", async (state) => {
    const client = new IAMClient({});
    const policy = await findPolicy(NAMES.instancePolicyName);

    if (!policy) {
      state.deletePolicyError = new Error(
        `Policy ${NAMES.instancePolicyName} not found.`
      );
    } else {
      return client.send(
        new DeletePolicyCommand({
          PolicyArn: policy.Arn,
        })
      );
    }
  })),
  new ScenarioOutput("deletePolicyResult", (state) => {
    if (state.deletePolicyError) {
      console.error(state.deletePolicyError);
      return MESSAGES.deletePolicyError.replace(
        "${INSTANCE_POLICY_NAME}",
        NAMES.instancePolicyName,
      );
    }
    return MESSAGES.deletedPolicy.replace(
      "${INSTANCE_POLICY_NAME}",
      NAMES.instancePolicyName,
    );
  })),
  new ScenarioAction("removeRoleFromInstanceProfile", async (state) => {
    try {
      const client = new IAMClient({});
      await client.send(
        new RemoveRoleFromInstanceProfileCommand({
```



```

        RoleName: NAMES.instanceRoleName,
        InstanceProfileName: NAMES.instanceProfileName,
    })),
    );
} catch (e) {
    state.removeRoleFromInstanceProfileError = e;
}
}),
new ScenarioOutput("removeRoleFromInstanceProfileResult", (state) => {
    if (state.removeRoleFromInstanceProfile) {
        console.error(state.removeRoleFromInstanceProfileError);
        return MESSAGES.removeRoleFromInstanceProfileError
            .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
            .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    }
    return MESSAGES.removedRoleFromInstanceProfile
        .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
}),
new ScenarioAction("deleteInstanceRole", async (state) => {
    try {
        const client = new IAMClient({});
        await client.send(
            new DeleteRoleCommand({
                RoleName: NAMES.instanceRoleName,
            })),
        );
    } catch (e) {
        state.deleteInstanceRoleError = e;
    }
}),
new ScenarioOutput("deleteInstanceRoleResult", (state) => {
    if (state.deleteInstanceRoleError) {
        console.error(state.deleteInstanceRoleError);
        return MESSAGES.deleteInstanceRoleError.replace(
            "${INSTANCE_ROLE_NAME}",
            NAMES.instanceRoleName,
        );
    }
    return MESSAGES.deletedInstanceRole.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
    );
}),

```

```
new ScenarioAction("deleteInstanceProfile", async (state) => {
  try {
    const client = new IAMClient({});
    await client.send(
      new DeleteInstanceProfileCommand({
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
  } catch (e) {
    state.deleteInstanceProfileError = e;
  }
}),
new ScenarioOutput("deleteInstanceProfileResult", (state) => {
  if (state.deleteInstanceProfileError) {
    console.error(state.deleteInstanceProfileError);
    return MESSAGES.deleteInstanceProfileError.replace(
      "${INSTANCE_PROFILE_NAME}",
      NAMES.instanceProfileName,
    );
  }
  return MESSAGES.deletedInstanceProfile.replace(
    "${INSTANCE_PROFILE_NAME}",
    NAMES.instanceProfileName,
  );
}),
new ScenarioAction("deleteAutoScalingGroup", async (state) => {
  try {
    await terminateGroupInstances(NAMES.autoScalingGroupName);
    await retry({ intervalInMs: 60000, maxRetries: 60 }, async () => {
      await deleteAutoScalingGroup(NAMES.autoScalingGroupName);
    });
  } catch (e) {
    state.deleteAutoScalingGroupError = e;
  }
}),
new ScenarioOutput("deleteAutoScalingGroupResult", (state) => {
  if (state.deleteAutoScalingGroupError) {
    console.error(state.deleteAutoScalingGroupError);
    return MESSAGES.deleteAutoScalingGroupError.replace(
      "${AUTO_SCALING_GROUP_NAME}",
      NAMES.autoScalingGroupName,
    );
  }
  return MESSAGES.deletedAutoScalingGroup.replace(
```

```

    "${AUTO_SCALING_GROUP_NAME}",
    NAMES.autoScalingGroupName,
  );
}),
new ScenarioAction("deleteLaunchTemplate", async (state) => {
  const client = new EC2Client({});
  try {
    await client.send(
      new DeleteLaunchTemplateCommand({
        LaunchTemplateName: NAMES.launchTemplateName,
      }),
    );
  } catch (e) {
    state.deleteLaunchTemplateError = e;
  }
}),
new ScenarioOutput("deleteLaunchTemplateResult", (state) => {
  if (state.deleteLaunchTemplateError) {
    console.error(state.deleteLaunchTemplateError);
    return MESSAGES.deleteLaunchTemplateError.replace(
      "${LAUNCH_TEMPLATE_NAME}",
      NAMES.launchTemplateName,
    );
  }
  return MESSAGES.deletedLaunchTemplate.replace(
    "${LAUNCH_TEMPLATE_NAME}",
    NAMES.launchTemplateName,
  );
}),
new ScenarioAction("deleteLoadBalancer", async (state) => {
  try {
    const client = new ElasticLoadBalancingV2Client({});
    const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
    await client.send(
      new DeleteLoadBalancerCommand({
        LoadBalancerArn: loadBalancer.LoadBalancerArn,
      }),
    );
    await retry({ intervalInMs: 1000, maxRetries: 60 }, async () => {
      const lb = await findLoadBalancer(NAMES.loadBalancerName);
      if (lb) {
        throw new Error("Load balancer still exists.");
      }
    });
  });
});

```

```

    } catch (e) {
      state.deleteLoadBalancerError = e;
    }
  )),
  new ScenarioOutput("deleteLoadBalancerResult", (state) => {
    if (state.deleteLoadBalancerError) {
      console.error(state.deleteLoadBalancerError);
      return MESSAGES.deleteLoadBalancerError.replace(
        "${LB_NAME}",
        NAMES.loadBalancerName,
      );
    }
    return MESSAGES.deletedLoadBalancer.replace(
      "${LB_NAME}",
      NAMES.loadBalancerName,
    );
  )),
  new ScenarioAction("deleteLoadBalancerTargetGroup", async (state) => {
    const client = new ElasticLoadBalancingV2Client({});
    try {
      const { TargetGroups } = await client.send(
        new DescribeTargetGroupsCommand({
          Names: [NAMES.loadBalancerTargetGroupName],
        }),
      );

      await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
        client.send(
          new DeleteTargetGroupCommand({
            TargetGroupArn: TargetGroups[0].TargetGroupArn,
          }),
        );
    } catch (e) {
      state.deleteLoadBalancerTargetGroupError = e;
    }
  )),
  new ScenarioOutput("deleteLoadBalancerTargetGroupResult", (state) => {
    if (state.deleteLoadBalancerTargetGroupError) {
      console.error(state.deleteLoadBalancerTargetGroupError);
      return MESSAGES.deleteLoadBalancerTargetGroupError.replace(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      );
    }
  })
);

```

```
}
return MESSAGES.deletedLoadBalancerTargetGroup.replace(
  "${TARGET_GROUP_NAME}",
  NAMES.loadBalancerTargetGroupName,
);
}),
new ScenarioAction("detachSsmOnlyRoleFromProfile", async (state) => {
  try {
    const client = new IAMClient({});
    await client.send(
      new RemoveRoleFromInstanceProfileCommand({
        InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
        RoleName: NAMES.ssmOnlyRoleName,
      }),
    );
  } catch (e) {
    state.detachSsmOnlyRoleFromProfileError = e;
  }
}),
new ScenarioOutput("detachSsmOnlyRoleFromProfileResult", (state) => {
  if (state.detachSsmOnlyRoleFromProfileError) {
    console.error(state.detachSsmOnlyRoleFromProfileError);
    return MESSAGES.detachSsmOnlyRoleFromProfileError
      .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
      .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
  }
  return MESSAGES.detachedSsmOnlyRoleFromProfile
    .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
    .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
}),
new ScenarioAction("detachSsmOnlyCustomRolePolicy", async (state) => {
  try {
    const iamClient = new IAMClient({});
    const ssmOnlyPolicy = await findPolicy(NAMES.ssmOnlyPolicyName);
    await iamClient.send(
      new DetachRolePolicyCommand({
        RoleName: NAMES.ssmOnlyRoleName,
        PolicyArn: ssmOnlyPolicy.Arn,
      }),
    );
  } catch (e) {
    state.detachSsmOnlyCustomRolePolicyError = e;
  }
}),
```

```
new ScenarioOutput("detachSsmOnlyCustomRolePolicyResult", (state) => {
  if (state.detachSsmOnlyCustomRolePolicyError) {
    console.error(state.detachSsmOnlyCustomRolePolicyError);
    return MESSAGES.detachSsmOnlyCustomRolePolicyError
      .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
      .replace("${POLICY_NAME}", NAMES.ssmOnlyPolicyName);
  }
  return MESSAGES.detachedSsmOnlyCustomRolePolicy
    .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
    .replace("${POLICY_NAME}", NAMES.ssmOnlyPolicyName);
}),
new ScenarioAction("detachSsmOnlyAWSRolePolicy", async (state) => {
  try {
    const iamClient = new IAMClient({});
    await iamClient.send(
      new DetachRolePolicyCommand({
        RoleName: NAMES.ssmOnlyRoleName,
        PolicyArn: "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore",
      }),
    );
  } catch (e) {
    state.detachSsmOnlyAWSRolePolicyError = e;
  }
}),
new ScenarioOutput("detachSsmOnlyAWSRolePolicyResult", (state) => {
  if (state.detachSsmOnlyAWSRolePolicyError) {
    console.error(state.detachSsmOnlyAWSRolePolicyError);
    return MESSAGES.detachSsmOnlyAWSRolePolicyError
      .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
      .replace("${POLICY_NAME}", "AmazonSSMManagedInstanceCore");
  }
  return MESSAGES.detachedSsmOnlyAWSRolePolicy
    .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
    .replace("${POLICY_NAME}", "AmazonSSMManagedInstanceCore");
}),
new ScenarioAction("deleteSsmOnlyInstanceProfile", async (state) => {
  try {
    const iamClient = new IAMClient({});
    await iamClient.send(
      new DeleteInstanceProfileCommand({
        InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
      }),
    );
  } catch (e) {
```

```
        state.deleteSsmOnlyInstanceProfileError = e;
    }
  })),
  new ScenarioOutput("deleteSsmOnlyInstanceProfileResult", (state) => {
    if (state.deleteSsmOnlyInstanceProfileError) {
      console.error(state.deleteSsmOnlyInstanceProfileError);
      return MESSAGES.deleteSsmOnlyInstanceProfileError.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.ssmOnlyInstanceProfileName,
      );
    }
    return MESSAGES.deletedSsmOnlyInstanceProfile.replace(
      "${INSTANCE_PROFILE_NAME}",
      NAMES.ssmOnlyInstanceProfileName,
    );
  })),
  new ScenarioAction("deleteSsmOnlyPolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
      const ssmOnlyPolicy = await findPolicy(NAMES.ssmOnlyPolicyName);
      await iamClient.send(
        new DeletePolicyCommand({
          PolicyArn: ssmOnlyPolicy.Arn,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyPolicyError = e;
    }
  })),
  new ScenarioOutput("deleteSsmOnlyPolicyResult", (state) => {
    if (state.deleteSsmOnlyPolicyError) {
      console.error(state.deleteSsmOnlyPolicyError);
      return MESSAGES.deleteSsmOnlyPolicyError.replace(
        "${POLICY_NAME}",
        NAMES.ssmOnlyPolicyName,
      );
    }
    return MESSAGES.deletedSsmOnlyPolicy.replace(
      "${POLICY_NAME}",
      NAMES.ssmOnlyPolicyName,
    );
  })),
  new ScenarioAction("deleteSsmOnlyRole", async (state) => {
    try {
```

```

    const iamClient = new IAMClient({});
    await iamClient.send(
      new DeleteRoleCommand({
        RoleName: NAMES.ssmOnlyRoleName,
      }),
    );
  } catch (e) {
    state.deleteSsmOnlyRoleError = e;
  }
}),
new ScenarioOutput("deleteSsmOnlyRoleResult", (state) => {
  if (state.deleteSsmOnlyRoleError) {
    console.error(state.deleteSsmOnlyRoleError);
    return MESSAGES.deleteSsmOnlyRoleError.replace(
      "${ROLE_NAME}",
      NAMES.ssmOnlyRoleName,
    );
  }
  return MESSAGES.deletedSsmOnlyRole.replace(
    "${ROLE_NAME}",
    NAMES.ssmOnlyRoleName,
  );
}),
new ScenarioAction(
  "revokeSecurityGroupIngress",
  async (
    /** @type {{ myIp: string, defaultSecurityGroup: { GroupId: string } }} */
    state,
  ) => {
    const ec2Client = new EC2Client({});

    try {
      await ec2Client.send(
        new RevokeSecurityGroupIngressCommand({
          GroupId: state.defaultSecurityGroup.GroupId,
          CidrIp: `${state.myIp}/32`,
          FromPort: 80,
          ToPort: 80,
          IpProtocol: "tcp",
        }),
      );
    } catch (e) {
      state.revokeSecurityGroupIngressError = e;
    }
  }

```



```

    },
  ),
  new ScenarioOutput("revokeSecurityGroupIngressResult", (state) => {
    if (state.revokeSecurityGroupIngressError) {
      console.error(state.revokeSecurityGroupIngressError);
      return MESSAGES.revokeSecurityGroupIngressError.replace(
        "${IP}",
        state.myIp,
      );
    }
    return MESSAGES.revokedSecurityGroupIngress.replace("${IP}", state.myIp);
  }),
];

/**
 * @param {string} policyName
 */
async function findPolicy(policyName) {
  const client = new IAMClient({});
  const paginatedPolicies = paginateListPolicies({ client }, {});
  for await (const page of paginatedPolicies) {
    const policy = page.Policies.find((p) => p.PolicyName === policyName);
    if (policy) {
      return policy;
    }
  }
}

/**
 * @param {string} groupName
 */
async function deleteAutoScalingGroup(groupName) {
  const client = new AutoScalingClient({});
  try {
    await client.send(
      new DeleteAutoScalingGroupCommand({
        AutoScalingGroupName: groupName,
      }),
    );
  } catch (err) {
    if (!(err instanceof Error)) {
      throw err;
    }
    console.log(err.name);
  }
}

```

```

        throw err;
    }
}

/**
 * @param {string} groupName
 */
async function terminateGroupInstances(groupName) {
    const autoScalingClient = new AutoScalingClient({});
    const group = await findAutoScalingGroup(groupName);
    await autoScalingClient.send(
        new UpdateAutoScalingGroupCommand({
            AutoScalingGroupName: group.AutoScalingGroupName,
            MinSize: 0,
        }),
    );
    for (const i of group.Instances) {
        await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
            autoScalingClient.send(
                new TerminateInstanceInAutoScalingGroupCommand({
                    InstanceId: i.InstanceId,
                    ShouldDecrementDesiredCapacity: true,
                }),
            ),
        );
    }
}

async function findAutoScalingGroup(groupName) {
    const client = new AutoScalingClient({});
    const paginatedGroups = paginateDescribeAutoScalingGroups({ client }, {});
    for await (const page of paginatedGroups) {
        const group = page.AutoScalingGroups.find(
            (g) => g.AutoScalingGroupName === groupName,
        );
        if (group) {
            return group;
        }
    }
    throw new Error(`Auto scaling group ${groupName} not found.`);
}

```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für JavaScript - API-Referenz.
 - [AttachLoadBalancerTargetGroups](#)
 - [CreateAutoScalingGroup](#)
 - [CreateInstanceProfile](#)
 - [CreateLaunchTemplate](#)
 - [CreateListener](#)
 - [CreateLoadBalancer](#)
 - [CreateTargetGroup](#)
 - [DeleteAutoScalingGroup](#)
 - [DeleteInstanceProfile](#)
 - [DeleteLaunchTemplate](#)
 - [DeleteLoadBalancer](#)
 - [DeleteTargetGroup](#)
 - [DescribeAutoScalingGroups](#)
 - [DescribeAvailabilityZones](#)
 - [DescribeIamInstanceProfileAssociations](#)
 - [DescribeInstances](#)
 - [DescribeLoadBalancers](#)
 - [DescribeSubnets](#)
 - [DescribeTargetGroups](#)
 - [DescribeTargetHealth](#)
 - [DescribeVpcs](#)
 - [RebootInstances](#)
 - [ReplacesIamInstanceProfileAssociation](#)
 - [TerminateInstanceInAutoScalingGroup](#)
 - [UpdateAutoScalingGroup](#)

Elastic Load Balancing — Beispiele für Version 2 mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie AWS SDK für JavaScript (v3) mit Elastic Load Balancing — Version 2 verwenden.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien anzeigen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb eines Service aufrufen oder mit anderen AWS-Services kombinieren.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kodex finden.

Themen

- [Erste Schritte](#)
- [Aktionen](#)
- [Szenarien](#)

Erste Schritte

Hello Elastic Load Balancing

Das folgende Codebeispiel zeigt, wie Sie mit Elastic Load Balancing beginnen können.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import {
  ElasticLoadBalancingV2Client,
  DescribeLoadBalancersCommand,
} from "@aws-sdk/client-elastic-load-balancing-v2";

export async function main() {
  const client = new ElasticLoadBalancingV2Client({});
  const { LoadBalancers } = await client.send(
    new DescribeLoadBalancersCommand({}),
  );
  const loadBalancersList = LoadBalancers.map(
    (lb) => `• ${lb.LoadBalancerName}: ${lb.DNSName}`,
  ).join("\n");
  console.log(
    "Hello, Elastic Load Balancing! Let's list some of your load balancers:\n",
    loadBalancersList,
  );
}

// Call function if run directly
import { fileURLToPath } from "node:url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main();
}
```

- Einzelheiten zur API finden Sie [DescribeLoadBalancers](#) in der AWS SDK für JavaScript API-Referenz.

Aktionen

CreateListener

Das folgende Codebeispiel zeigt die Verwendung `CreateListener`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
const client = new ElasticLoadBalancingV2Client({});
const { Listeners } = await client.send(
  new CreateListenerCommand({
    LoadBalancerArn: state.loadBalancerArn,
    Protocol: state.targetGroupProtocol,
    Port: state.targetGroupPort,
    DefaultActions: [
      { Type: "forward", TargetGroupArn: state.targetGroupArn },
    ],
  }),
);
```

- Einzelheiten zur API finden Sie [CreateListener](#) in der AWS SDK für JavaScript API-Referenz.

CreateLoadBalancer

Das folgende Codebeispiel zeigt die Verwendung `CreateLoadBalancer`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
const client = new ElasticLoadBalancingV2Client({});
const { LoadBalancers } = await client.send(
  new CreateLoadBalancerCommand({
    Name: NAMES.loadBalancerName,
    Subnets: state.subnets,
  }),
);
state.loadBalancerDns = LoadBalancers[0].DNSName;
state.loadBalancerArn = LoadBalancers[0].LoadBalancerArn;
await waitUntilLoadBalancerAvailable(
  { client },
  { Names: [NAMES.loadBalancerName] },
);
```

- Einzelheiten zur API finden Sie [CreateLoadBalancer](#) in der AWS SDK für JavaScript API-Referenz.

CreateTargetGroup

Das folgende Codebeispiel zeigt die Verwendung `CreateTargetGroup`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
const client = new ElasticLoadBalancingV2Client({});
const { TargetGroups } = await client.send(
  new CreateTargetGroupCommand({
    Name: NAMES.loadBalancerTargetGroupName,
    Protocol: "HTTP",
    Port: 80,
    HealthCheckPath: "/healthcheck",
    HealthCheckIntervalSeconds: 10,
    HealthCheckTimeoutSeconds: 5,
    HealthyThresholdCount: 2,
    UnhealthyThresholdCount: 2,
    VpcId: state.defaultVpc,
  }),
);
```

- Einzelheiten zur API finden Sie [CreateTargetGroup](#) in der AWS SDK für JavaScript API-Referenz.

DeleteLoadBalancer

Das folgende Codebeispiel zeigt die Verwendung `DeleteLoadBalancer`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
const client = new ElasticLoadBalancingV2Client({});
const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
await client.send(
  new DeleteLoadBalancerCommand({
    LoadBalancerArn: loadBalancer.LoadBalancerArn,
  }),
);
await retry({ intervalInMs: 1000, maxRetries: 60 }, async () => {
  const lb = await findLoadBalancer(NAMES.loadBalancerName);
  if (lb) {
    throw new Error("Load balancer still exists.");
  }
});
```

- Einzelheiten zur API finden Sie [DeleteLoadBalancer](#) in der AWS SDK für JavaScript API-Referenz.

DeleteTargetGroup

Das folgende Codebeispiel zeigt die Verwendung `DeleteTargetGroup`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
const client = new ElasticLoadBalancingV2Client({});
```



```
try {
  const { TargetGroups } = await client.send(
    new DescribeTargetGroupsCommand({
      Names: [NAMES.loadBalancerTargetGroupName],
    }),
  );

  await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
    client.send(
      new DeleteTargetGroupCommand({
        TargetGroupArn: TargetGroups[0].TargetGroupArn,
      }),
    ),
  );
} catch (e) {
  state.deleteLoadBalancerTargetGroupError = e;
}
```

- Einzelheiten zur API finden Sie [DeleteTargetGroup](#) in der AWS SDK für JavaScript API-Referenz.

DescribeLoadBalancers

Das folgende Codebeispiel zeigt die Verwendung `DescribeLoadBalancers`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import {
  ElasticLoadBalancingV2Client,
  DescribeLoadBalancersCommand,
} from "@aws-sdk/client-elastic-load-balancing-v2";

export async function main() {
```

```
const client = new ElasticLoadBalancingV2Client({});
const { LoadBalancers } = await client.send(
  new DescribeLoadBalancersCommand({}),
);
const loadBalancersList = LoadBalancers.map(
  (lb) => `• ${lb.LoadBalancerName}: ${lb.DNSName}`,
).join("\n");
console.log(
  "Hello, Elastic Load Balancing! Let's list some of your load balancers:\n",
  loadBalancersList,
);
}

// Call function if run directly
import { fileURLToPath } from "node:url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main();
}
```

- Einzelheiten zur API finden Sie [DescribeLoadBalancers](#) in der AWS SDK für JavaScript API-Referenz.

DescribeTargetGroups

Das folgende Codebeispiel zeigt die Verwendung `DescribeTargetGroups`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
const client = new ElasticLoadBalancingV2Client({});
const { TargetGroups } = await client.send(
  new DescribeTargetGroupsCommand({
    Names: [NAMES.loadBalancerTargetGroupName],
  }),
);
```

- Einzelheiten zur API finden Sie [DescribeTargetGroups](#) in der AWS SDK für JavaScript API-Referenz.

DescribeTargetHealth

Das folgende Codebeispiel zeigt die Verwendung `DescribeTargetHealth`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
const { TargetHealthDescriptions } = await client.send(
  new DescribeTargetHealthCommand({
    TargetGroupArn: TargetGroups[0].TargetGroupArn,
  }),
);
```

- Einzelheiten zur API finden Sie [DescribeTargetHealth](#) in der AWS SDK für JavaScript API-Referenz.

Szenarien

Erstellen und Verwalten eines ausfallsicheren Services

Das folgende Codebeispiel zeigt, wie Sie einen Webservice mit Load Balancing erstellen, der Buch-, Film- und Liedempfehlungen zurückgibt. Das Beispiel zeigt, wie der Service auf Fehler reagiert und wie der Service für mehr Ausfallsicherheit umstrukturiert werden kann.

- Verwenden Sie eine Amazon EC2 Auto Scaling Scaling-Gruppe, um Amazon Elastic Compute Cloud (Amazon EC2) -Instances auf der Grundlage einer Startvorlage zu erstellen und die Anzahl der Instances in einem bestimmten Bereich zu halten.
- Verarbeiten und verteilen Sie HTTP-Anfragen mit Elastic Load Balancing.

- Überwachen Sie den Zustand von Instances in einer Auto-Scaling-Gruppe und leiten Sie Anfragen nur an fehlerfreie Instances weiter.
- Führen Sie auf jeder EC2 Instanz einen Python-Webserver aus, um HTTP-Anfragen zu verarbeiten. Der Webserver reagiert mit Empfehlungen und Zustandsprüfungen.
- Simulieren Sie einen Empfehlungsservice mit einer Amazon DynamoDB-Tabelle.
- Steuern Sie die Antwort des Webserver auf Anfragen und Zustandsprüfungen, indem Sie die AWS Systems Manager Parameter aktualisieren.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Führen Sie ein interaktives Szenario an einer Eingabeaufforderung aus.

```
#!/usr/bin/env node
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import {
  Scenario,
  parseScenarioArgs,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * The workflow steps are split into three stages:
 *   - deploy
 *   - demo
 *   - destroy
 *
 * Each of these stages has a corresponding file prefixed with steps-*.
 */
import { deploySteps } from "./steps-deploy.js";
import { demoSteps } from "./steps-demo.js";
import { destroySteps } from "./steps-destroy.js";

/**
```

```

* The context is passed to every scenario. Scenario steps
* will modify the context.
*/
const context = {};

/**
 * Three Scenarios are created for the workflow. A Scenario is an orchestration
 * class
 * that simplifies running a series of steps.
 */
export const scenarios = {
  // Deploys all resources necessary for the workflow.
  deploy: new Scenario("Resilient Workflow - Deploy", deploySteps, context),
  // Demonstrates how a fragile web service can be made more resilient.
  demo: new Scenario("Resilient Workflow - Demo", demoSteps, context),
  // Destroys the resources created for the workflow.
  destroy: new Scenario("Resilient Workflow - Destroy", destroySteps, context),
};

// Call function if run directly
import { fileURLToPath } from "node:url";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios, {
    name: "Resilient Workflow",
    synopsis:
      "node index.js --scenario <deploy | demo | destroy> [-h|--help] [-y|--yes] [-v|--verbose]",
    description: "Deploy and interact with scalable EC2 instances.",
  });
}

```

Erstellen Sie Schritte, um alle Ressourcen bereitzustellen.

```

import { join } from "node:path";
import { readFileSync, writeFileSync } from "node:fs";
import axios from "axios";

import {
  BatchWriteItemCommand,
  CreateTableCommand,
  DynamoDBClient,

```

```
    waitUntilTableExists,
  } from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  CreateKeyPairCommand,
  CreateLaunchTemplateCommand,
  DescribeAvailabilityZonesCommand,
  DescribeVpcsCommand,
  DescribeSubnetsCommand,
  DescribeSecurityGroupsCommand,
  AuthorizeSecurityGroupIngressCommand,
} from "@aws-sdk/client-ec2";
import {
  IAMClient,
  CreatePolicyCommand,
  CreateRoleCommand,
  CreateInstanceProfileCommand,
  AddRoleToInstanceProfileCommand,
  AttachRolePolicyCommand,
  waitUntilInstanceProfileExists,
} from "@aws-sdk/client-iam";
import { SSMClient, GetParameterCommand } from "@aws-sdk/client-ssm";
import {
  CreateAutoScalingGroupCommand,
  AutoScalingClient,
  AttachLoadBalancerTargetGroupsCommand,
} from "@aws-sdk/client-auto-scaling";
import {
  CreateListenerCommand,
  CreateLoadBalancerCommand,
  CreateTargetGroupCommand,
  ElasticLoadBalancingV2Client,
  waitUntilLoadBalancerAvailable,
} from "@aws-sdk/client-elastic-load-balancing-v2";

import {
  ScenarioOutput,
  ScenarioInput,
  ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { saveState } from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES, RESOURCES_PATH, ROOT } from "../constants.js";
```

```

import { initParamsSteps } from "../steps-reset-params.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const deploySteps = [
  new ScenarioOutput("introduction", MESSAGES.introduction, { header: true }),
  new ScenarioInput("confirmDeployment", MESSAGES.confirmDeployment, {
    type: "confirm",
  }),
  new ScenarioAction(
    "handleConfirmDeployment",
    (c) => c.confirmDeployment === false && process.exit(),
  ),
  new ScenarioOutput(
    "creatingTable",
    MESSAGES.creatingTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioAction("createTable", async () => {
    const client = new DynamoDBClient({});
    await client.send(
      new CreateTableCommand({
        TableName: NAMES.tableName,
        ProvisionedThroughput: {
          ReadCapacityUnits: 5,
          WriteCapacityUnits: 5,
        },
        AttributeDefinitions: [
          {
            AttributeName: "MediaType",
            AttributeType: "S",
          },
          {
            AttributeName: "ItemId",
            AttributeType: "N",
          },
        ],
        KeySchema: [
          {
            AttributeName: "MediaType",
            KeyType: "HASH",
          },
          {
            AttributeName: "ItemId",

```

```

        KeyType: "RANGE",
      },
    ],
  )),
);
await waitUntilTableExists({ client }, { TableName: NAMES.tableName });
}),
new ScenarioOutput(
  "createdTable",
  MESSAGES.createdTable.replace("${TABLE_NAME}", NAMES.tableName),
),
new ScenarioOutput(
  "populatingTable",
  MESSAGES.populatingTable.replace("${TABLE_NAME}", NAMES.tableName),
),
new ScenarioAction("populateTable", () => {
  const client = new DynamoDBClient({});
  /**
   * @type {{ default: import("@aws-sdk/client-dynamodb").PutRequest['Item'][] }}
   */
  const recommendations = JSON.parse(
    readFileSync(join(RESOURCES_PATH, "recommendations.json")),
  );

  return client.send(
    new BatchWriteItemCommand({
      RequestItems: {
        [NAMES.tableName]: recommendations.map((item) => ({
          PutRequest: { Item: item },
        })),
      },
    }),
  );
}),
new ScenarioOutput(
  "populatedTable",
  MESSAGES.populatedTable.replace("${TABLE_NAME}", NAMES.tableName),
),
new ScenarioOutput(
  "creatingKeyPair",
  MESSAGES.creatingKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
),
new ScenarioAction("createKeyPair", async () => {
  const client = new EC2Client({});

```



```
const { KeyMaterial } = await client.send(
  new CreateKeyPairCommand({
    KeyName: NAMES.keyPairName,
  }),
);

writeFileSync(`${NAMES.keyPairName}.pem`, KeyMaterial, { mode: 0o600 });
}),
new ScenarioOutput(
  "createdKeyPair",
  MESSAGES.createdKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
),
new ScenarioOutput(
  "creatingInstancePolicy",
  MESSAGES.creatingInstancePolicy.replace(
    "${INSTANCE_POLICY_NAME}",
    NAMES.instancePolicyName,
  ),
),
new ScenarioAction("createInstancePolicy", async (state) => {
  const client = new IAMClient({});
  const {
    Policy: { Arn },
  } = await client.send(
    new CreatePolicyCommand({
      PolicyName: NAMES.instancePolicyName,
      PolicyDocument: readFileSync(
        join(RESOURCES_PATH, "instance_policy.json"),
      ),
    }),
  ),
  state.instancePolicyArn = Arn;
}),
new ScenarioOutput("createdInstancePolicy", (state) =>
  MESSAGES.createdInstancePolicy
    .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
    .replace("${INSTANCE_POLICY_ARN}", state.instancePolicyArn),
),
new ScenarioOutput(
  "creatingInstanceRole",
  MESSAGES.creatingInstanceRole.replace(
    "${INSTANCE_ROLE_NAME}",
    NAMES.instanceRoleName,
  ),
),
```

```

    ),
    new ScenarioAction("createInstanceRole", () => {
      const client = new IAMClient({});
      return client.send(
        new CreateRoleCommand({
          RoleName: NAMES.instanceRoleName,
          AssumeRolePolicyDocument: readFileSync(
            join(ROOT, "assume-role-policy.json"),
          ),
        }),
      ),
    }),
  );
}),
new ScenarioOutput(
  "createdInstanceRole",
  MESSAGES.createdInstanceRole.replace(
    "${INSTANCE_ROLE_NAME}",
    NAMES.instanceRoleName,
  ),
),
new ScenarioOutput(
  "attachingPolicyToRole",
  MESSAGES.attachingPolicyToRole
    .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName)
    .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName),
),
new ScenarioAction("attachPolicyToRole", async (state) => {
  const client = new IAMClient({});
  await client.send(
    new AttachRolePolicyCommand({
      RoleName: NAMES.instanceRoleName,
      PolicyArn: state.instancePolicyArn,
    }),
  ),
);
}),
new ScenarioOutput(
  "attachedPolicyToRole",
  MESSAGES.attachedPolicyToRole
    .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
    .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
),
new ScenarioOutput(
  "creatingInstanceProfile",
  MESSAGES.creatingInstanceProfile.replace(
    "${INSTANCE_PROFILE_NAME}",

```

```

    NAMES.instanceProfileName,
  ),
),
new ScenarioAction("createInstanceProfile", async (state) => {
  const client = new IAMClient({});
  const {
    InstanceProfile: { Arn },
  } = await client.send(
    new CreateInstanceProfileCommand({
      InstanceProfileName: NAMES.instanceProfileName,
    }),
  );
  state.instanceProfileArn = Arn;

  await waitUntilInstanceProfileExists(
    { client },
    { InstanceProfileName: NAMES.instanceProfileName },
  );
}),
new ScenarioOutput("createdInstanceProfile", (state) =>
  MESSAGES.createdInstanceProfile
    .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
    .replace("${INSTANCE_PROFILE_ARN}", state.instanceProfileArn),
),
new ScenarioOutput(
  "addingRoleToInstanceProfile",
  MESSAGES.addingRoleToInstanceProfile
    .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
    .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
),
new ScenarioAction("addRoleToInstanceProfile", () => {
  const client = new IAMClient({});
  return client.send(
    new AddRoleToInstanceProfileCommand({
      RoleName: NAMES.instanceRoleName,
      InstanceProfileName: NAMES.instanceProfileName,
    }),
  );
}),
new ScenarioOutput(
  "addedRoleToInstanceProfile",
  MESSAGES.addedRoleToInstanceProfile
    .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
    .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),

```

```

    ),
    ...initParamsSteps,
    new ScenarioOutput("creatingLaunchTemplate", MESSAGES.creatingLaunchTemplate),
    new ScenarioAction("createLaunchTemplate", async () => {
      const ssmClient = new SSMClient({});
      const { Parameter } = await ssmClient.send(
        new GetParameterCommand({
          Name: "/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2",
        }),
      );
      const ec2Client = new EC2Client({});
      await ec2Client.send(
        new CreateLaunchTemplateCommand({
          LaunchTemplateName: NAMES.launchTemplateName,
          LaunchTemplateData: {
            InstanceType: "t3.micro",
            ImageId: Parameter.Value,
            IamInstanceProfile: { Name: NAMES.instanceProfileName },
            UserData: readFileSync(
              join(RESOURCES_PATH, "server_startup_script.sh"),
            ).toString("base64"),
            KeyName: NAMES.keyPairName,
          },
        }),
      );
    }),
    new ScenarioOutput(
      "createdLaunchTemplate",
      MESSAGES.createdLaunchTemplate.replace(
        "${LAUNCH_TEMPLATE_NAME}",
        NAMES.launchTemplateName,
      ),
    ),
    new ScenarioOutput(
      "creatingAutoScalingGroup",
      MESSAGES.creatingAutoScalingGroup.replace(
        "${AUTO_SCALING_GROUP_NAME}",
        NAMES.autoScalingGroupName,
      ),
    ),
    new ScenarioAction("createAutoScalingGroup", async (state) => {
      const ec2Client = new EC2Client({});
      const { AvailabilityZones } = await ec2Client.send(
        new DescribeAvailabilityZonesCommand({}),
      );
    })
  ],
);

```

```

    );
    state.availabilityZoneNames = AvailabilityZones.map((az) => az.ZoneName);
    const autoScalingClient = new AutoScalingClient({});
    await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
      autoScalingClient.send(
        new CreateAutoScalingGroupCommand({
          AvailabilityZones: state.availabilityZoneNames,
          AutoScalingGroupName: NAMES.autoScalingGroupName,
          LaunchTemplate: {
            LaunchTemplateName: NAMES.launchTemplateName,
            Version: "$Default",
          },
          MinSize: 3,
          MaxSize: 3,
        }),
      ),
    );
  })),
  new ScenarioOutput(
    "createdAutoScalingGroup",
    /**
     * @param {{ availabilityZoneNames: string[] }} state
     */
    (state) =>
      MESSAGES.createdAutoScalingGroup
        .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName)
        .replace(
          "${AVAILABILITY_ZONE_NAMES}",
          state.availabilityZoneNames.join(", "),
        ),
  ),
  new ScenarioInput("confirmContinue", MESSAGES.confirmContinue, {
    type: "confirm",
  }),
  new ScenarioOutput("loadBalancer", MESSAGES.loadBalancer),
  new ScenarioOutput("gettingVpc", MESSAGES.gettingVpc),
  new ScenarioAction("getVpc", async (state) => {
    const client = new EC2Client({});
    const { Vpcs } = await client.send(
      new DescribeVpcsCommand({
        Filters: [{ Name: "is-default", Values: ["true"] }],
      }),
    );
  });
  state.defaultVpc = Vpcs[0].VpcId;

```

```

    })),
    new ScenarioOutput("gotVpc", (state) =>
      MESSAGES.gotVpc.replace("${VPC_ID}", state.defaultVpc),
    ),
    new ScenarioOutput("gettingSubnets", MESSAGES.gettingSubnets),
    new ScenarioAction("getSubnets", async (state) => {
      const client = new EC2Client({});
      const { Subnets } = await client.send(
        new DescribeSubnetsCommand({
          Filters: [
            { Name: "vpc-id", Values: [state.defaultVpc] },
            { Name: "availability-zone", Values: state.availabilityZoneNames },
            { Name: "default-for-az", Values: ["true"] },
          ],
        })
      );
      state.subnets = Subnets.map((subnet) => subnet.SubnetId);
    }),
    new ScenarioOutput(
      "gotSubnets",
      /**
       * @param {{ subnets: string[] }} state
       */
      (state) =>
        MESSAGES.gotSubnets.replace("${SUBNETS}", state.subnets.join(", ")),
    ),
    new ScenarioOutput(
      "creatingLoadBalancerTargetGroup",
      MESSAGES.creatingLoadBalancerTargetGroup.replace(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      ),
    ),
    new ScenarioAction("createLoadBalancerTargetGroup", async (state) => {
      const client = new ElasticLoadBalancingV2Client({});
      const { TargetGroups } = await client.send(
        new CreateTargetGroupCommand({
          Name: NAMES.loadBalancerTargetGroupName,
          Protocol: "HTTP",
          Port: 80,
          HealthCheckPath: "/healthcheck",
          HealthCheckIntervalSeconds: 10,
          HealthCheckTimeoutSeconds: 5,
          HealthyThresholdCount: 2,

```

```

        UnhealthyThresholdCount: 2,
        VpcId: state.defaultVpc,
    )),
    );
    const targetGroup = TargetGroups[0];
    state.targetGroupArn = targetGroup.TargetGroupArn;
    state.targetGroupProtocol = targetGroup.Protocol;
    state.targetGroupPort = targetGroup.Port;
  )),
  new ScenarioOutput(
    "createdLoadBalancerTargetGroup",
    MESSAGES.createdLoadBalancerTargetGroup.replace(
      "${TARGET_GROUP_NAME}",
      NAMES.loadBalancerTargetGroupName,
    ),
  ),
  new ScenarioOutput(
    "creatingLoadBalancer",
    MESSAGES.creatingLoadBalancer.replace("${LB_NAME}", NAMES.loadBalancerName),
  ),
  new ScenarioAction("createLoadBalancer", async (state) => {
    const client = new ElasticLoadBalancingV2Client({});
    const { LoadBalancers } = await client.send(
      new CreateLoadBalancerCommand({
        Name: NAMES.loadBalancerName,
        Subnets: state.subnets,
      }),
    );
    state.loadBalancerDns = LoadBalancers[0].DNSName;
    state.loadBalancerArn = LoadBalancers[0].LoadBalancerArn;
    await waitUntilLoadBalancerAvailable(
      { client },
      { Names: [NAMES.loadBalancerName] },
    );
  )),
  new ScenarioOutput("createdLoadBalancer", (state) =>
    MESSAGES.createdLoadBalancer
      .replace("${LB_NAME}", NAMES.loadBalancerName)
      .replace("${DNS_NAME}", state.loadBalancerDns),
  ),
  new ScenarioOutput(
    "creatingListener",
    MESSAGES.creatingLoadBalancerListener
      .replace("${LB_NAME}", NAMES.loadBalancerName)

```

```

        .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName),
    ),
    new ScenarioAction("createListener", async (state) => {
        const client = new ElasticLoadBalancingV2Client({});
        const { Listeners } = await client.send(
            new CreateListenerCommand({
                LoadBalancerArn: state.loadBalancerArn,
                Protocol: state.targetGroupProtocol,
                Port: state.targetGroupPort,
                DefaultActions: [
                    { Type: "forward", TargetGroupArn: state.targetGroupArn },
                ],
            })
        );
        const listener = Listeners[0];
        state.loadBalancerListenerArn = listener.ListenerArn;
    }),
    new ScenarioOutput("createdListener", (state) =>
        MESSAGES.createdLoadBalancerListener.replace(
            "${LB_LISTENER_ARN}",
            state.loadBalancerListenerArn,
        ),
    ),
    new ScenarioOutput(
        "attachingLoadBalancerTargetGroup",
        MESSAGES.attachingLoadBalancerTargetGroup
            .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName)
            .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName),
    ),
    new ScenarioAction("attachLoadBalancerTargetGroup", async (state) => {
        const client = new AutoScalingClient({});
        await client.send(
            new AttachLoadBalancerTargetGroupsCommand({
                AutoScalingGroupName: NAMES.autoScalingGroupName,
                TargetGroupARNs: [state.targetGroupArn],
            })
        );
    }),
    new ScenarioOutput(
        "attachedLoadBalancerTargetGroup",
        MESSAGES.attachedLoadBalancerTargetGroup,
    ),
    new ScenarioOutput("verifyingInboundPort", MESSAGES.verifyingInboundPort),
    new ScenarioAction(

```



```

    "verifyInboundPort",
    /**
     *
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-ec2').SecurityGroup}}
state
    */
    async (state) => {
        const client = new EC2Client({});
        const { SecurityGroups } = await client.send(
            new DescribeSecurityGroupsCommand({
                Filters: [{ Name: "group-name", Values: ["default"] }],
            }),
        );
        if (!SecurityGroups) {
            state.verifyInboundPortError = new Error(MESSAGES.noSecurityGroups);
        }
        state.defaultSecurityGroup = SecurityGroups[0];

        /**
         * @type {string}
         */
        const ipResponse = (await axios.get("http://checkip.amazonaws.com")).data;
        state.myIp = ipResponse.trim();
        const myIpRules = state.defaultSecurityGroup.IpPermissions.filter(
            ({ IpRanges }) =>
                IpRanges.some(
                    ({ CidrIp }) =>
                        CidrIp.startsWith(state.myIp) || CidrIp === "0.0.0.0/0",
                ),
        )
            .filter(({ IpProtocol }) => IpProtocol === "tcp")
            .filter(({ FromPort }) => FromPort === 80);

        state.myIpRules = myIpRules;
    },
),
new ScenarioOutput(
    "verifiedInboundPort",
    /**
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
        if (state.myIpRules.length > 0) {
            return MESSAGES.foundIpRules.replace(

```

```

        "${IP_RULES}",
        JSON.stringify(state.myIpRules, null, 2),
    );
}
return MESSAGES.noIpRules;
},
),
new ScenarioInput(
    "shouldAddInboundRule",
    /**
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
        if (state.myIpRules.length > 0) {
            return false;
        }
        return MESSAGES.noIpRules;
    },
    { type: "confirm" },
),
new ScenarioAction(
    "addInboundRule",
    /**
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-ec2').SecurityGroup }} state
     */
    async (state) => {
        if (!state.shouldAddInboundRule) {
            return;
        }

        const client = new EC2Client({});
        await client.send(
            new AuthorizeSecurityGroupIngressCommand({
                GroupId: state.defaultSecurityGroup.GroupId,
                CidrIp: `${state.myIp}/32`,
                FromPort: 80,
                ToPort: 80,
                IpProtocol: "tcp",
            })
        );
    },
),
new ScenarioOutput("addedInboundRule", (state) => {

```

```

    if (state.shouldAddInboundRule) {
      return MESSAGES.addedInboundRule.replace("${IP_ADDRESS}", state.myIp);
    }
    return false;
  })),
  new ScenarioOutput("verifyingEndpoint", (state) =>
    MESSAGES.verifyingEndpoint.replace("${DNS_NAME}", state.loadBalancerDns),
  ),
  new ScenarioAction("verifyEndpoint", async (state) => {
    try {
      const response = await retry({ intervalInMs: 2000, maxRetries: 30 }, () =>
        axios.get(`http://${state.loadBalancerDns}`),
      );
      state.endpointResponse = JSON.stringify(response.data, null, 2);
    } catch (e) {
      state.verifyEndpointError = e;
    }
  })),
  new ScenarioOutput("verifiedEndpoint", (state) => {
    if (state.verifyEndpointError) {
      console.error(state.verifyEndpointError);
    } else {
      return MESSAGES.verifiedEndpoint.replace(
        "${ENDPOINT_RESPONSE}",
        state.endpointResponse,
      );
    }
  })),
  saveState,
];

```

Erstellen Sie Schritte, um die Demo auszuführen.

```

import { readFileSync } from "node:fs";
import { join } from "node:path";

import axios from "axios";

import {
  DescribeTargetGroupsCommand,
  DescribeTargetHealthCommand,
  ElasticLoadBalancingV2Client,

```

```
} from "@aws-sdk/client-elastic-load-balancing-v2";
import {
  DescribeInstanceInformationCommand,
  PutParameterCommand,
  SSMClient,
  SendCommandCommand,
} from "@aws-sdk/client-ssm";
import {
  IAMClient,
  CreatePolicyCommand,
  CreateRoleCommand,
  AttachRolePolicyCommand,
  CreateInstanceProfileCommand,
  AddRoleToInstanceProfileCommand,
  waitUntilInstanceProfileExists,
} from "@aws-sdk/client-iam";
import {
  AutoScalingClient,
  DescribeAutoScalingGroupsCommand,
  TerminateInstanceInAutoScalingGroupCommand,
} from "@aws-sdk/client-auto-scaling";
import {
  DescribeIamInstanceProfileAssociationsCommand,
  EC2Client,
  RebootInstancesCommand,
  ReplaceIamInstanceProfileAssociationCommand,
} from "@aws-sdk/client-ec2";

import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/scenario.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES, RESOURCES_PATH } from "../constants.js";
import { findLoadBalancer } from "../shared.js";

const getRecommendation = new ScenarioAction(
  "getRecommendation",
  async (state) => {
    const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
    if (loadBalancer) {
      state.loadBalancerDnsName = loadBalancer.DNSName;
    }
  }
);
```

```

    try {
      state.recommendation = (
        await axios.get(`http://${state.loadBalancerDnsName}`)
      ).data;
    } catch (e) {
      state.recommendation = e instanceof Error ? e.message : e;
    }
  } else {
    throw new Error(MESSAGES.demoFindLoadBalancerError);
  }
},
);

const getRecommendationResult = new ScenarioOutput(
  "getRecommendationResult",
  (state) =>
    `Recommendation:\n${JSON.stringify(state.recommendation, null, 2)}`,
  { preformatted: true },
);

const getHealthCheck = new ScenarioAction("getHealthCheck", async (state) => {
  const client = new ElasticLoadBalancingV2Client({});
  const { TargetGroups } = await client.send(
    new DescribeTargetGroupsCommand({
      Names: [NAMES.loadBalancerTargetGroupName],
    }),
  );
});

const { TargetHealthDescriptions } = await client.send(
  new DescribeTargetHealthCommand({
    TargetGroupArn: TargetGroups[0].TargetGroupArn,
  }),
);
state.targetHealthDescriptions = TargetHealthDescriptions;
});

const getHealthCheckResult = new ScenarioOutput(
  "getHealthCheckResult",
  /**
   * @param {{ targetHealthDescriptions: import('@aws-sdk/client-elastic-load-
balancing-v2').TargetHealthDescription[] }} state
   */
  (state) => {
    const status = state.targetHealthDescriptions

```

```
    .map((th) => `${th.Target.Id}: ${th.TargetHealth.State}`)
    .join("\n");
    return `Health check:\n${status}`;
  },
  { preformatted: true },
);

const loadBalancerLoop = new ScenarioAction(
  "loadBalancerLoop",
  getRecommendation.action,
  {
    whileConfig: {
      whileFn: ({ loadBalancerCheck }) => loadBalancerCheck,
      input: new ScenarioInput(
        "loadBalancerCheck",
        MESSAGES.demoLoadBalancerCheck,
        {
          type: "confirm",
        },
      ),
      output: getRecommendationResult,
    },
  },
);

const healthCheckLoop = new ScenarioAction(
  "healthCheckLoop",
  getHealthCheck.action,
  {
    whileConfig: {
      whileFn: ({ healthCheck }) => healthCheck,
      input: new ScenarioInput("healthCheck", MESSAGES.demoHealthCheck, {
        type: "confirm",
      }),
      output: getHealthCheckResult,
    },
  },
);

const statusSteps = [
  getRecommendation,
  getRecommendationResult,
  getHealthCheck,
  getHealthCheckResult,
```

```
];

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const demoSteps = [
  new ScenarioOutput("header", MESSAGES.demoHeader, { header: true }),
  new ScenarioOutput("sanityCheck", MESSAGES.demoSanityCheck),
  ...statusSteps,
  new ScenarioInput(
    "brokenDependencyConfirmation",
    MESSAGES.demoBrokenDependencyConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("brokenDependency", async (state) => {
    if (!state.brokenDependencyConfirmation) {
      process.exit();
    } else {
      const client = new SSMClient({});
      state.badTableName = `fake-table-${Date.now()}`;
      await client.send(
        new PutParameterCommand({
          Name: NAMES.ssmTableNameKey,
          Value: state.badTableName,
          Overwrite: true,
          Type: "String",
        }),
      );
    }
  }),
  new ScenarioOutput("testBrokenDependency", (state) =>
    MESSAGES.demoTestBrokenDependency.replace(
      "${TABLE_NAME}",
      state.badTableName,
    ),
  ),
  ...statusSteps,
  new ScenarioInput(
    "staticResponseConfirmation",
    MESSAGES.demoStaticResponseConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("staticResponse", async (state) => {
    if (!state.staticResponseConfirmation) {
```

```

        process.exit();
    } else {
        const client = new SSMClient({});
        await client.send(
            new PutParameterCommand({
                Name: NAMES.ssmFailureResponseKey,
                Value: "static",
                Overwrite: true,
                Type: "String",
            }),
        );
    }
}),
new ScenarioOutput("testStaticResponse", MESSAGES.demoTestStaticResponse),
...statusSteps,
new ScenarioInput(
    "badCredentialsConfirmation",
    MESSAGES.demoBadCredentialsConfirmation,
    { type: "confirm" },
),
new ScenarioAction("badCredentialsExit", (state) => {
    if (!state.badCredentialsConfirmation) {
        process.exit();
    }
}),
new ScenarioAction("fixDynamoDBName", async () => {
    const client = new SSMClient({});
    await client.send(
        new PutParameterCommand({
            Name: NAMES.ssmTableNameKey,
            Value: NAMES.tableName,
            Overwrite: true,
            Type: "String",
        }),
    );
}),
new ScenarioAction(
    "badCredentials",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-auto-scaling').Instance }}
state
    */
    async (state) => {
        await createSsmOnlyInstanceProfile();
    }
)

```



```
const autoScalingClient = new AutoScalingClient({});
const { AutoScalingGroups } = await autoScalingClient.send(
  new DescribeAutoScalingGroupsCommand({
    AutoScalingGroupNames: [NAMES.autoScalingGroupName],
  }),
);
state.targetInstance = AutoScalingGroups[0].Instances[0];
const ec2Client = new EC2Client({});
const { IamInstanceProfileAssociations } = await ec2Client.send(
  new DescribeIamInstanceProfileAssociationsCommand({
    Filters: [
      { Name: "instance-id", Values: [state.targetInstance.InstanceId] },
    ],
  }),
);
state.instanceProfileAssociationId =
  IamInstanceProfileAssociations[0].AssociationId;
await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
  ec2Client.send(
    new ReplaceIamInstanceProfileAssociationCommand({
      AssociationId: state.instanceProfileAssociationId,
      IamInstanceProfile: { Name: NAMES.ssmOnlyInstanceProfileName },
    }),
  ),
);

await ec2Client.send(
  new RebootInstancesCommand({
    InstanceIds: [state.targetInstance.InstanceId],
  }),
);

const ssmClient = new SSMClient({});
await retry({ intervalInMs: 20000, maxRetries: 15 }, async () => {
  const { InstanceInformationList } = await ssmClient.send(
    new DescribeInstanceInformationCommand({}),
  );

  const instance = InstanceInformationList.find(
    (info) => info.InstanceId === state.targetInstance.InstanceId,
  );

  if (!instance) {
    throw new Error("Instance not found.");
  }
});
```

```

    }
  });

  await ssmClient.send(
    new SendCommandCommand({
      InstanceIds: [state.targetInstance.InstanceId],
      DocumentName: "AWS-RunShellScript",
      Parameters: { commands: ["cd / && sudo python3 server.py 80"] },
    }),
  );
},
),
new ScenarioOutput(
  "testBadCredentials",
  /**
   * @param {{ targetInstance: import('@aws-sdk/client-ssm').InstanceInformation}}
state
  */
  (state) =>
    MESSAGES.demoTestBadCredentials.replace(
      "${INSTANCE_ID}",
      state.targetInstance.InstanceId,
    ),
),
loadBalancerLoop,
new ScenarioInput(
  "deepHealthCheckConfirmation",
  MESSAGES.demoDeepHealthCheckConfirmation,
  { type: "confirm" },
),
new ScenarioAction("deepHealthCheckExit", (state) => {
  if (!state.deepHealthCheckConfirmation) {
    process.exit();
  }
}),
new ScenarioAction("deepHealthCheck", async () => {
  const client = new SSMClient({});
  await client.send(
    new PutParameterCommand({
      Name: NAMES.ssmHealthCheckKey,
      Value: "deep",
      Overwrite: true,
      Type: "String",
    }),
  ),

```

```

    );
  }),
  new ScenarioOutput("testDeepHealthCheck", MESSAGES.demoTestDeepHealthCheck),
  healthCheckLoop,
  loadBalancerLoop,
  new ScenarioInput(
    "killInstanceConfirmation",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-
    ssm').InstanceInformation }} state
     */
    (state) =>
      MESSAGES.demoKillInstanceConfirmation.replace(
        "${INSTANCE_ID}",
        state.targetInstance.InstanceId,
      ),
    { type: "confirm" },
  ),
  new ScenarioAction("killInstanceExit", (state) => {
    if (!state.killInstanceConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction(
    "killInstance",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-
    ssm').InstanceInformation }} state
     */
    async (state) => {
      const client = new AutoScalingClient({});
      await client.send(
        new TerminateInstanceInAutoScalingGroupCommand({
          InstanceId: state.targetInstance.InstanceId,
          ShouldDecrementDesiredCapacity: false,
        }),
      );
    },
  ),
  new ScenarioOutput("testKillInstance", MESSAGES.demoTestKillInstance),
  healthCheckLoop,
  loadBalancerLoop,
  new ScenarioInput("failOpenConfirmation", MESSAGES.demoFailOpenConfirmation, {
    type: "confirm",
  }

```

```

    })),
    new ScenarioAction("failOpenExit", (state) => {
      if (!state.failOpenConfirmation) {
        process.exit();
      }
    })),
    new ScenarioAction("failOpen", () => {
      const client = new SSMClient({});
      return client.send(
        new PutParameterCommand({
          Name: NAMES.ssmTableNameKey,
          Value: `fake-table-${Date.now()}`,
          Overwrite: true,
          Type: "String",
        })
      ),
    );
  })),
  new ScenarioOutput("testFailOpen", MESSAGES.demoFailOpenTest),
  healthCheckLoop,
  loadBalancerLoop,
  new ScenarioInput(
    "resetTableConfirmation",
    MESSAGES.demoResetTableConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("resetTableExit", (state) => {
    if (!state.resetTableConfirmation) {
      process.exit();
    }
  })),
  new ScenarioAction("resetTable", async () => {
    const client = new SSMClient({});
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: NAMES.tableName,
        Overwrite: true,
        Type: "String",
      })
    ),
  );
  })),
  new ScenarioOutput("testResetTable", MESSAGES.demoTestResetTable),
  healthCheckLoop,
  loadBalancerLoop,

```

```
];

async function createSsmOnlyInstanceProfile() {
  const iamClient = new IAMClient({});
  const { Policy } = await iamClient.send(
    new CreatePolicyCommand({
      PolicyName: NAMES.ssmOnlyPolicyName,
      PolicyDocument: readFileSync(
        join(RESOURCES_PATH, "ssm_only_policy.json"),
      ),
    }),
  );
  await iamClient.send(
    new CreateRoleCommand({
      RoleName: NAMES.ssmOnlyRoleName,
      AssumeRolePolicyDocument: JSON.stringify({
        Version: "2012-10-17",
        Statement: [
          {
            Effect: "Allow",
            Principal: { Service: "ec2.amazonaws.com" },
            Action: "sts:AssumeRole",
          },
        ],
      }),
    }),
  );
  await iamClient.send(
    new AttachRolePolicyCommand({
      RoleName: NAMES.ssmOnlyRoleName,
      PolicyArn: Policy.Arn,
    }),
  );
  await iamClient.send(
    new AttachRolePolicyCommand({
      RoleName: NAMES.ssmOnlyRoleName,
      PolicyArn: "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore",
    }),
  );
  const { InstanceProfile } = await iamClient.send(
    new CreateInstanceProfileCommand({
      InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
    }),
  );
};
```

```
await waitUntilInstanceProfileExists(
  { client: iamClient },
  { InstanceProfileName: NAMES.ssmOnlyInstanceProfileName },
);
await iamClient.send(
  new AddRoleToInstanceProfileCommand({
    InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
    RoleName: NAMES.ssmOnlyRoleName,
  }),
);

return InstanceProfile;
}
```

Erstellen Sie Schritte, um alle Ressourcen zu vernichten.

```
import { unlinkSync } from "node:fs";

import { DynamoDBClient, DeleteTableCommand } from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  DeleteKeyPairCommand,
  DeleteLaunchTemplateCommand,
  RevokeSecurityGroupIngressCommand,
} from "@aws-sdk/client-ec2";
import {
  IAMClient,
  DeleteInstanceProfileCommand,
  RemoveRoleFromInstanceProfileCommand,
  DeletePolicyCommand,
  DeleteRoleCommand,
  DetachRolePolicyCommand,
  paginateListPolicies,
} from "@aws-sdk/client-iam";
import {
  AutoScalingClient,
  DeleteAutoScalingGroupCommand,
  TerminateInstanceInAutoScalingGroupCommand,
  UpdateAutoScalingGroupCommand,
  paginateDescribeAutoScalingGroups,
} from "@aws-sdk/client-auto-scaling";
import {
```

```

    DeleteLoadBalancerCommand,
    DeleteTargetGroupCommand,
    DescribeTargetGroupsCommand,
    ElasticLoadBalancingV2Client,
  } from "@aws-sdk/client-elastic-load-balancing-v2";

import {
  ScenarioOutput,
  ScenarioInput,
  ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { loadState } from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES } from "../constants.js";
import { findLoadBalancer } from "../shared.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const destroySteps = [
  loadState,
  new ScenarioInput("destroy", MESSAGES.destroy, { type: "confirm" }),
  new ScenarioAction(
    "abort",
    (state) => state.destroy === false && process.exit(),
  ),
  new ScenarioAction("deleteTable", async (c) => {
    try {
      const client = new DynamoDBClient({});
      await client.send(new DeleteTableCommand({ TableName: NAMES.tableName }));
    } catch (e) {
      c.deleteTableError = e;
    }
  }),
  new ScenarioOutput("deleteTableResult", (state) => {
    if (state.deleteTableError) {
      console.error(state.deleteTableError);
      return MESSAGES.deleteTableError.replace(
        "${TABLE_NAME}",
        NAMES.tableName,
      );
    }
    return MESSAGES.deletedTable.replace("${TABLE_NAME}", NAMES.tableName);
  })
];

```

```
    }},
    new ScenarioAction("deleteKeyPair", async (state) => {
      try {
        const client = new EC2Client({});
        await client.send(
          new DeleteKeyPairCommand({ KeyName: NAMES.keyPairName }),
        );
        unlinkSync(`${NAMES.keyPairName}.pem`);
      } catch (e) {
        state.deleteKeyPairError = e;
      }
    }),
    new ScenarioOutput("deleteKeyPairResult", (state) => {
      if (state.deleteKeyPairError) {
        console.error(state.deleteKeyPairError);
        return MESSAGES.deleteKeyPairError.replace(
          "${KEY_PAIR_NAME}",
          NAMES.keyPairName,
        );
      }
      return MESSAGES.deletedKeyPair.replace(
        "${KEY_PAIR_NAME}",
        NAMES.keyPairName,
      );
    }),
    new ScenarioAction("detachPolicyFromRole", async (state) => {
      try {
        const client = new IAMClient({});
        const policy = await findPolicy(NAMES.instancePolicyName);

        if (!policy) {
          state.detachPolicyFromRoleError = new Error(
            `Policy ${NAMES.instancePolicyName} not found.`,
          );
        } else {
          await client.send(
            new DetachRolePolicyCommand({
              RoleName: NAMES.instanceRoleName,
              PolicyArn: policy.Arn,
            }),
          );
        }
      } catch (e) {
        state.detachPolicyFromRoleError = e;
      }
    })
  ],
  state,
```



```
    }
  }},
  new ScenarioOutput("detachedPolicyFromRole", (state) => {
    if (state.detachPolicyFromRoleError) {
      console.error(state.detachPolicyFromRoleError);
      return MESSAGES.detachPolicyFromRoleError
        .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    }
    return MESSAGES.detachedPolicyFromRole
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
  }},
  new ScenarioAction("deleteInstancePolicy", async (state) => {
    const client = new IAMClient({});
    const policy = await findPolicy(NAMES.instancePolicyName);

    if (!policy) {
      state.deletePolicyError = new Error(
        `Policy ${NAMES.instancePolicyName} not found.`
      );
    } else {
      return client.send(
        new DeletePolicyCommand({
          PolicyArn: policy.Arn,
        }),
      );
    }
  }},
  new ScenarioOutput("deletePolicyResult", (state) => {
    if (state.deletePolicyError) {
      console.error(state.deletePolicyError);
      return MESSAGES.deletePolicyError.replace(
        "${INSTANCE_POLICY_NAME}",
        NAMES.instancePolicyName,
      );
    }
    return MESSAGES.deletedPolicy.replace(
      "${INSTANCE_POLICY_NAME}",
      NAMES.instancePolicyName,
    );
  }},
  new ScenarioAction("removeRoleFromInstanceProfile", async (state) => {
    try {
```

```

    const client = new IAMClient({});
    await client.send(
      new RemoveRoleFromInstanceProfileCommand({
        RoleName: NAMES.instanceRoleName,
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
  } catch (e) {
    state.removeRoleFromInstanceProfileError = e;
  }
}),
new ScenarioOutput("removeRoleFromInstanceProfileResult", (state) => {
  if (state.removeRoleFromInstanceProfile) {
    console.error(state.removeRoleFromInstanceProfileError);
    return MESSAGES.removeRoleFromInstanceProfileError
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
  }
  return MESSAGES.removedRoleFromInstanceProfile
    .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
    .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
}),
new ScenarioAction("deleteInstanceRole", async (state) => {
  try {
    const client = new IAMClient({});
    await client.send(
      new DeleteRoleCommand({
        RoleName: NAMES.instanceRoleName,
      }),
    );
  } catch (e) {
    state.deleteInstanceRoleError = e;
  }
}),
new ScenarioOutput("deleteInstanceRoleResult", (state) => {
  if (state.deleteInstanceRoleError) {
    console.error(state.deleteInstanceRoleError);
    return MESSAGES.deleteInstanceRoleError.replace(
      "${INSTANCE_ROLE_NAME}",
      NAMES.instanceRoleName,
    );
  }
  return MESSAGES.deletedInstanceRole.replace(
    "${INSTANCE_ROLE_NAME}",

```

```

    NAMES.instanceRoleName,
  );
}),
new ScenarioAction("deleteInstanceProfile", async (state) => {
  try {
    const client = new IAMClient({});
    await client.send(
      new DeleteInstanceProfileCommand({
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
  } catch (e) {
    state.deleteInstanceProfileError = e;
  }
}),
new ScenarioOutput("deleteInstanceProfileResult", (state) => {
  if (state.deleteInstanceProfileError) {
    console.error(state.deleteInstanceProfileError);
    return MESSAGES.deleteInstanceProfileError.replace(
      "${INSTANCE_PROFILE_NAME}",
      NAMES.instanceProfileName,
    );
  }
  return MESSAGES.deletedInstanceProfile.replace(
    "${INSTANCE_PROFILE_NAME}",
    NAMES.instanceProfileName,
  );
}),
new ScenarioAction("deleteAutoScalingGroup", async (state) => {
  try {
    await terminateGroupInstances(NAMES.autoScalingGroupName);
    await retry({ intervalInMs: 60000, maxRetries: 60 }, async () => {
      await deleteAutoScalingGroup(NAMES.autoScalingGroupName);
    });
  } catch (e) {
    state.deleteAutoScalingGroupError = e;
  }
}),
new ScenarioOutput("deleteAutoScalingGroupResult", (state) => {
  if (state.deleteAutoScalingGroupError) {
    console.error(state.deleteAutoScalingGroupError);
    return MESSAGES.deleteAutoScalingGroupError.replace(
      "${AUTO_SCALING_GROUP_NAME}",
      NAMES.autoScalingGroupName,
    );
  }
  return MESSAGES.deletedAutoScalingGroup.replace(
    "${AUTO_SCALING_GROUP_NAME}",
    NAMES.autoScalingGroupName,
  );
}),

```

```

    );
  }
  return MESSAGES.deletedAutoScalingGroup.replace(
    "${AUTO_SCALING_GROUP_NAME}",
    NAMES.autoScalingGroupName,
  );
}),
new ScenarioAction("deleteLaunchTemplate", async (state) => {
  const client = new EC2Client({});
  try {
    await client.send(
      new DeleteLaunchTemplateCommand({
        LaunchTemplateName: NAMES.launchTemplateName,
      }),
    );
  } catch (e) {
    state.deleteLaunchTemplateError = e;
  }
}),
new ScenarioOutput("deleteLaunchTemplateResult", (state) => {
  if (state.deleteLaunchTemplateError) {
    console.error(state.deleteLaunchTemplateError);
    return MESSAGES.deleteLaunchTemplateError.replace(
      "${LAUNCH_TEMPLATE_NAME}",
      NAMES.launchTemplateName,
    );
  }
  return MESSAGES.deletedLaunchTemplate.replace(
    "${LAUNCH_TEMPLATE_NAME}",
    NAMES.launchTemplateName,
  );
}),
new ScenarioAction("deleteLoadBalancer", async (state) => {
  try {
    const client = new ElasticLoadBalancingV2Client({});
    const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
    await client.send(
      new DeleteLoadBalancerCommand({
        LoadBalancerArn: loadBalancer.LoadBalancerArn,
      }),
    );
    await retry({ intervalInMs: 1000, maxRetries: 60 }, async () => {
      const lb = await findLoadBalancer(NAMES.loadBalancerName);
      if (lb) {

```

```

        throw new Error("Load balancer still exists.");
    }
});
} catch (e) {
    state.deleteLoadBalancerError = e;
}
}),
new ScenarioOutput("deleteLoadBalancerResult", (state) => {
    if (state.deleteLoadBalancerError) {
        console.error(state.deleteLoadBalancerError);
        return MESSAGES.deleteLoadBalancerError.replace(
            "${LB_NAME}",
            NAMES.loadBalancerName,
        );
    }
    return MESSAGES.deletedLoadBalancer.replace(
        "${LB_NAME}",
        NAMES.loadBalancerName,
    );
}),
new ScenarioAction("deleteLoadBalancerTargetGroup", async (state) => {
    const client = new ElasticLoadBalancingV2Client({});
    try {
        const { TargetGroups } = await client.send(
            new DescribeTargetGroupsCommand({
                Names: [NAMES.loadBalancerTargetGroupName],
            }),
        );

        await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
            client.send(
                new DeleteTargetGroupCommand({
                    TargetGroupArn: TargetGroups[0].TargetGroupArn,
                }),
            ),
        );
    } catch (e) {
        state.deleteLoadBalancerTargetGroupError = e;
    }
}),
new ScenarioOutput("deleteLoadBalancerTargetGroupResult", (state) => {
    if (state.deleteLoadBalancerTargetGroupError) {
        console.error(state.deleteLoadBalancerTargetGroupError);
        return MESSAGES.deleteLoadBalancerTargetGroupError.replace(

```

```
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
    );
}
return MESSAGES.deletedLoadBalancerTargetGroup.replace(
    "${TARGET_GROUP_NAME}",
    NAMES.loadBalancerTargetGroupName,
);
}),
new ScenarioAction("detachSsmOnlyRoleFromProfile", async (state) => {
    try {
        const client = new IAMClient({});
        await client.send(
            new RemoveRoleFromInstanceProfileCommand({
                InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
                RoleName: NAMES.ssmOnlyRoleName,
            }),
        );
    } catch (e) {
        state.detachSsmOnlyRoleFromProfileError = e;
    }
}),
new ScenarioOutput("detachSsmOnlyRoleFromProfileResult", (state) => {
    if (state.detachSsmOnlyRoleFromProfileError) {
        console.error(state.detachSsmOnlyRoleFromProfileError);
        return MESSAGES.detachSsmOnlyRoleFromProfileError
            .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
            .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
    }
    return MESSAGES.detachedSsmOnlyRoleFromProfile
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
}),
new ScenarioAction("detachSsmOnlyCustomRolePolicy", async (state) => {
    try {
        const iamClient = new IAMClient({});
        const ssmOnlyPolicy = await findPolicy(NAMES.ssmOnlyPolicyName);
        await iamClient.send(
            new DetachRolePolicyCommand({
                RoleName: NAMES.ssmOnlyRoleName,
                PolicyArn: ssmOnlyPolicy.Arn,
            }),
        );
    } catch (e) {
```

```

        state.detachSsmOnlyCustomRolePolicyError = e;
    }
  })),
  new ScenarioOutput("detachSsmOnlyCustomRolePolicyResult", (state) => {
    if (state.detachSsmOnlyCustomRolePolicyError) {
      console.error(state.detachSsmOnlyCustomRolePolicyError);
      return MESSAGES.detachSsmOnlyCustomRolePolicyError
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", NAMES.ssmOnlyPolicyName);
    }
    return MESSAGES.detachedSsmOnlyCustomRolePolicy
      .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
      .replace("${POLICY_NAME}", NAMES.ssmOnlyPolicyName);
  })),
  new ScenarioAction("detachSsmOnlyAWSRolePolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
      await iamClient.send(
        new DetachRolePolicyCommand({
          RoleName: NAMES.ssmOnlyRoleName,
          PolicyArn: "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore",
        }),
      );
    } catch (e) {
      state.detachSsmOnlyAWSRolePolicyError = e;
    }
  })),
  new ScenarioOutput("detachSsmOnlyAWSRolePolicyResult", (state) => {
    if (state.detachSsmOnlyAWSRolePolicyError) {
      console.error(state.detachSsmOnlyAWSRolePolicyError);
      return MESSAGES.detachSsmOnlyAWSRolePolicyError
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", "AmazonSSMManagedInstanceCore");
    }
    return MESSAGES.detachedSsmOnlyAWSRolePolicy
      .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
      .replace("${POLICY_NAME}", "AmazonSSMManagedInstanceCore");
  })),
  new ScenarioAction("deleteSsmOnlyInstanceProfile", async (state) => {
    try {
      const iamClient = new IAMClient({});
      await iamClient.send(
        new DeleteInstanceProfileCommand({
          InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,

```

```

    }},
    );
  } catch (e) {
    state.deleteSsmOnlyInstanceProfileError = e;
  }
}),
new ScenarioOutput("deleteSsmOnlyInstanceProfileResult", (state) => {
  if (state.deleteSsmOnlyInstanceProfileError) {
    console.error(state.deleteSsmOnlyInstanceProfileError);
    return MESSAGES.deleteSsmOnlyInstanceProfileError.replace(
      "${INSTANCE_PROFILE_NAME}",
      NAMES.ssmOnlyInstanceProfileName,
    );
  }
  return MESSAGES.deletedSsmOnlyInstanceProfile.replace(
    "${INSTANCE_PROFILE_NAME}",
    NAMES.ssmOnlyInstanceProfileName,
  );
}),
new ScenarioAction("deleteSsmOnlyPolicy", async (state) => {
  try {
    const iamClient = new IAMClient({});
    const ssmOnlyPolicy = await findPolicy(NAMES.ssmOnlyPolicyName);
    await iamClient.send(
      new DeletePolicyCommand({
        PolicyArn: ssmOnlyPolicy.Arn,
      }),
    );
  } catch (e) {
    state.deleteSsmOnlyPolicyError = e;
  }
}),
new ScenarioOutput("deleteSsmOnlyPolicyResult", (state) => {
  if (state.deleteSsmOnlyPolicyError) {
    console.error(state.deleteSsmOnlyPolicyError);
    return MESSAGES.deleteSsmOnlyPolicyError.replace(
      "${POLICY_NAME}",
      NAMES.ssmOnlyPolicyName,
    );
  }
  return MESSAGES.deletedSsmOnlyPolicy.replace(
    "${POLICY_NAME}",
    NAMES.ssmOnlyPolicyName,
  );
});

```



```

    })),
    new ScenarioAction("deleteSsmOnlyRole", async (state) => {
      try {
        const iamClient = new IAMClient({});
        await iamClient.send(
          new DeleteRoleCommand({
            RoleName: NAMES.ssmOnlyRoleName,
          }),
        );
      } catch (e) {
        state.deleteSsmOnlyRoleError = e;
      }
    })),
    new ScenarioOutput("deleteSsmOnlyRoleResult", (state) => {
      if (state.deleteSsmOnlyRoleError) {
        console.error(state.deleteSsmOnlyRoleError);
        return MESSAGES.deleteSsmOnlyRoleError.replace(
          "${ROLE_NAME}",
          NAMES.ssmOnlyRoleName,
        );
      }
      return MESSAGES.deletedSsmOnlyRole.replace(
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
      );
    })),
    new ScenarioAction(
      "revokeSecurityGroupIngress",
      async (
        /** @type {{ myIp: string, defaultSecurityGroup: { GroupId: string } }} */
        state,
      ) => {
        const ec2Client = new EC2Client({});

        try {
          await ec2Client.send(
            new RevokeSecurityGroupIngressCommand({
              GroupId: state.defaultSecurityGroup.GroupId,
              CidrIp: `${state.myIp}/32`,
              FromPort: 80,
              ToPort: 80,
              IpProtocol: "tcp",
            }),
          );
        }
      }
    ),
  ],
);

```

```

        } catch (e) {
            state.revokeSecurityGroupIngressError = e;
        }
    },
),
new ScenarioOutput("revokeSecurityGroupIngressResult", (state) => {
    if (state.revokeSecurityGroupIngressError) {
        console.error(state.revokeSecurityGroupIngressError);
        return MESSAGES.revokeSecurityGroupIngressError.replace(
            "${IP}",
            state.myIp,
        );
    }
    return MESSAGES.revokedSecurityGroupIngress.replace("${IP}", state.myIp);
}),
];

/**
 * @param {string} policyName
 */
async function findPolicy(policyName) {
    const client = new IAMClient({});
    const paginatedPolicies = paginateListPolicies({ client }, {});
    for await (const page of paginatedPolicies) {
        const policy = page.Policies.find((p) => p.PolicyName === policyName);
        if (policy) {
            return policy;
        }
    }
}

/**
 * @param {string} groupName
 */
async function deleteAutoScalingGroup(groupName) {
    const client = new AutoScalingClient({});
    try {
        await client.send(
            new DeleteAutoScalingGroupCommand({
                AutoScalingGroupName: groupName,
            }),
        );
    } catch (err) {
        if (!(err instanceof Error)) {

```

```

        throw err;
    }
    console.log(err.name);
    throw err;
}
}

/**
 * @param {string} groupName
 */
async function terminateGroupInstances(groupName) {
    const autoScalingClient = new AutoScalingClient({});
    const group = await findAutoScalingGroup(groupName);
    await autoScalingClient.send(
        new UpdateAutoScalingGroupCommand({
            AutoScalingGroupName: group.AutoScalingGroupName,
            MinSize: 0,
        }),
    );
    for (const i of group.Instances) {
        await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
            autoScalingClient.send(
                new TerminateInstanceInAutoScalingGroupCommand({
                    InstanceId: i.InstanceId,
                    ShouldDecrementDesiredCapacity: true,
                }),
            ),
    );
    }
}

async function findAutoScalingGroup(groupName) {
    const client = new AutoScalingClient({});
    const paginatedGroups = paginateDescribeAutoScalingGroups({ client }, {});
    for await (const page of paginatedGroups) {
        const group = page.AutoScalingGroups.find(
            (g) => g.AutoScalingGroupName === groupName,
        );
        if (group) {
            return group;
        }
    }
    throw new Error(`Auto scaling group ${groupName} not found.`);
}

```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für JavaScript - API-Referenz.
 - [AttachLoadBalancerTargetGroups](#)
 - [CreateAutoScalingGroup](#)
 - [CreateInstanceProfile](#)
 - [CreateLaunchTemplate](#)
 - [CreateListener](#)
 - [CreateLoadBalancer](#)
 - [CreateTargetGroup](#)
 - [DeleteAutoScalingGroup](#)
 - [DeleteInstanceProfile](#)
 - [DeleteLaunchTemplate](#)
 - [DeleteLoadBalancer](#)
 - [DeleteTargetGroup](#)
 - [DescribeAutoScalingGroups](#)
 - [DescribeAvailabilityZones](#)
 - [DescribeIamInstanceProfileAssociations](#)
 - [DescribeInstances](#)
 - [DescribeLoadBalancers](#)
 - [DescribeSubnets](#)
 - [DescribeTargetGroups](#)
 - [DescribeTargetHealth](#)
 - [DescribeVpcs](#)
 - [RebootInstances](#)
 - [ReplaceIamInstanceProfileAssociation](#)
 - [TerminateInstanceInAutoScalingGroup](#)
 - [UpdateAutoScalingGroup](#)

AWS Entity Resolution Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie AWS SDK für JavaScript (v3) mit verwenden AWS Entity Resolution.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kodex finden.

Themen

- [Erste Schritte](#)
- [Aktionen](#)

Erste Schritte

Hallo AWS Entity Resolution

Das folgende Codebeispiel veranschaulicht, wie Sie mit der Verwendung von AWS Entity Resolution beginnen.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { fileURLToPath } from "node:url";
import {
  EntityResolutionClient,
  ListMatchingWorkflowsCommand,
} from "@aws-sdk/client-entityresolution";

export const main = async () => {
  const region = "eu-west-1";
  const erClient = new EntityResolutionClient({ region: region });
```

```
try {
  const command = new ListMatchingWorkflowsCommand({});
  const response = await erClient.send(command);
  const workflowSummaries = response.workflowSummaries;
  for (const workflowSummary of workflowSummaries) {
    console.log(`Attribute name: ${workflowSummaries[0].workflowName} `);
  }
  if (workflowSummaries.length === 0) {
    console.log("No matching workflows found.");
  }
} catch (error) {
  console.error(
    `An error occurred in listing the workflow summaries: ${error.message} \n
    Exiting program.`
  );
  return;
};
```

- Einzelheiten zur API finden Sie [ListMatchingWorkflows](#) in der AWS SDK für JavaScript API-Referenz.

Aktionen

CreateMatchingWorkflow

Das folgende Codebeispiel zeigt die Verwendung `CreateMatchingWorkflow`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
//The default inputs for this demo are read from the ../inputs.json.
```

```
import { fileURLToPath } from "node:url";

import {
  CreateMatchingWorkflowCommand,
  EntityResolutionClient,
} from "@aws-sdk/client-entityresolution";
import data from "../inputs.json" with { type: "json" };

const region = "eu-west-1";
const erClient = new EntityResolutionClient({ region: region });

export const main = async () => {
  const createMatchingWorkflowParams = {
    roleArn: `${data.inputs.roleArn}`,
    workflowName: `${data.inputs.workflowName}`,
    description: "Created by using the AWS SDK for JavaScript (v3).",
    inputSourceConfig: [
      {
        inputSourceARN: `${data.inputs.JSONinputSourceARN}`,
        schemaName: `${data.inputs.schemaNameJson}`,
        applyNormalization: false,
      },
      {
        inputSourceARN: `${data.inputs.CSVinputSourceARN}`,
        schemaName: `${data.inputs.schemaNameCSV}`,
        applyNormalization: false,
      },
    ],
    outputSourceConfig: [
      {
        outputS3Path: `s3://${data.inputs.myBucketName}/eroutput`,
        output: [
          {
            name: "id",
          },
          {
            name: "name",
          },
          {
            name: "email",
          },
          {
            name: "phone",
          },
        ],
      },
    ],
  };
  const command = new CreateMatchingWorkflowCommand(createMatchingWorkflowParams);
  const response = await erClient.send(command);
  console.log(response);
}
```

```
    ],
    applyNormalization: false,
  },
],
resolutionTechniques: { resolutionType: "ML_MATCHING" },
};
try {
  const command = new CreateMatchingWorkflowCommand(
    createMatchingWorkflowParams,
  );
  const response = await erClient.send(command);

  console.log(
    `Workflow created successfully.\n The workflow ARN is:
    ${response.workflowArn}`,
  );
} catch (caught) {
  console.error(caught.message);
  throw caught;
}
};
```

- Einzelheiten zur API finden Sie [CreateMatchingWorkflow](#) in der AWS SDK für JavaScript API-Referenz.

CreateSchemaMapping

Das folgende Codebeispiel zeigt die Verwendung `CreateSchemaMapping`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
//The default inputs for this demo are read from the ../inputs.json.
```



```
import { fileURLToPath } from "node:url";

import {
  CreateSchemaMappingCommand,
  EntityResolutionClient,
} from "@aws-sdk/client-entityresolution";
import data from "../inputs.json" with { type: "json" };

const region = "eu-west-1";
const erClient = new EntityResolutionClient({ region: region });

export const main = async () => {
  const createSchemaMappingParamsJson = {
    schemaName: `${data.inputs.schemaNameJson}`,
    mappedInputFields: [
      {
        fieldName: "id",
        type: "UNIQUE_ID",
      },
      {
        fieldName: "name",
        type: "NAME",
      },
      {
        fieldName: "email",
        type: "EMAIL_ADDRESS",
      },
    ],
  };
  const createSchemaMappingParamsCSV = {
    schemaName: `${data.inputs.schemaNameCSV}`,
    mappedInputFields: [
      {
        fieldName: "id",
        type: "UNIQUE_ID",
      },
      {
        fieldName: "name",
        type: "NAME",
      },
      {
        fieldName: "email",
        type: "EMAIL_ADDRESS",
      },
    ],
  };
}
```

```
    },
    {
      fieldName: "phone",
      type: "PROVIDER_ID",
      subType: "STRING",
    },
  ],
};
try {
  const command = new CreateSchemaMappingCommand(
    createSchemaMappingParamsJson,
  );
  const response = await erClient.send(command);
  console.log("The JSON schema mapping name is ", response.schemaName);
} catch (error) {
  console.log("error ", error.message);
}
};
```

- Einzelheiten zur API finden Sie [CreateSchemaMapping](#) in der AWS SDK für JavaScript API-Referenz.

DeleteMatchingWorkflow

Das folgende Codebeispiel zeigt die Verwendung `DeleteMatchingWorkflow`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
//The default inputs for this demo are read from the ../inputs.json.

import { fileURLToPath } from "node:url";
```

```
import {
  DeleteMatchingWorkflowCommand,
  EntityResolutionClient,
} from "@aws-sdk/client-entityresolution";
import data from "../inputs.json" with { type: "json" };

const region = "eu-west-1";
const erClient = new EntityResolutionClient({ region: region });

export const main = async () => {
  try {
    const deleteWorkflowParams = {
      workflowName: `${data.inputs.workflowName}`,
    };
    const command = new DeleteMatchingWorkflowCommand(deleteWorkflowParams);
    const response = await erClient.send(command);
    console.log("Workflow deleted successfully!", response);
  } catch (error) {
    console.log("error ", error);
  }
};
```

- Einzelheiten zur API finden Sie [DeleteMatchingWorkflow](#) in der AWS SDK für JavaScript API-Referenz.

DeleteSchemaMapping

Das folgende Codebeispiel zeigt die Verwendung `DeleteSchemaMapping`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
//The default inputs for this demo are read from the ../inputs.json.
```

```
import { fileURLToPath } from "node:url";

import {
  DeleteSchemaMappingCommand,
  EntityResolutionClient,
} from "@aws-sdk/client-entityresolution";
import data from "../inputs.json" with { type: "json" };

const region = "eu-west-1";
const erClient = new EntityResolutionClient({ region: region });

export const main = async () => {
  const deleteSchemaMapping = {
    schemaName: `${data.inputs.schemaNameJson}`,
  };
  try {
    const command = new DeleteSchemaMappingCommand(deleteSchemaMapping);
    const response = await erClient.send(command);
    console.log("Schema mapping deleted successfully. ", response);
  } catch (error) {
    console.log("error ", error);
  }
};
```

- Einzelheiten zur API finden Sie [DeleteSchemaMapping](#) in der AWS SDK für JavaScript API-Referenz.

GetMatchingJob

Das folgende Codebeispiel zeigt die Verwendung `GetMatchingJob`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
//The default inputs for this demo are read from the ../inputs.json.

import { fileURLToPath } from "node:url";

import {
  GetMatchingJobCommand,
  EntityResolutionClient,
} from "@aws-sdk/client-entityresolution";
import data from "../inputs.json" with { type: "json" };

const region = "eu-west-1";
const erClient = new EntityResolutionClient({ region: region });

export const main = async () => {
  async function getInfo() {
    const getJobInfoParams = {
      workflowName: `${data.inputs.workflowName}`,
      jobId: `${data.inputs.jobId}`,
    };
    try {
      const command = new GetMatchingJobCommand(getJobInfoParams);
      const response = await erClient.send(command);
      console.log(`Job status: ${response.status}`);
    } catch (error) {
      console.log("error ", error.message);
    }
  }
};
```

- Einzelheiten zur API finden Sie [GetMatchingJob](#) in der AWS SDK für JavaScript API-Referenz.

GetSchemaMapping

Das folgende Codebeispiel zeigt die Verwendung `GetSchemaMapping`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
//The default inputs for this demo are read from the ../inputs.json.

import { fileURLToPath } from "node:url";

import {
  GetSchemaMappingCommand,
  EntityResolutionClient,
} from "@aws-sdk/client-entityresolution";
import data from "../inputs.json" with { type: "json" };

const region = "eu-west-1";
const erClient = new EntityResolutionClient({ region: region });

export const main = async () => {
  const getSchemaMappingJsonParams = {
    schemaName: `${data.inputs.schemaNameJson}`,
  };
  try {
    const command = new GetSchemaMappingCommand(getSchemaMappingJsonParams);
    const response = await erClient.send(command);
    console.log(response);
    console.log(
      `Schema mapping for the JSON data:\n ${response.mappedInputFields[0]}`,
    );
    console.log("Schema mapping ARN is: ", response.schemaArn);
  } catch (caught) {
    console.error(caught.message);
    throw caught;
  }
};
```

- Einzelheiten zur API finden Sie [GetSchemaMapping](#) in der AWS SDK für JavaScript API-Referenz.

ListSchemaMappings

Das folgende Codebeispiel zeigt die Verwendung `ListSchemaMappings`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
//The default inputs for this demo are read from the ../inputs.json.

import { fileURLToPath } from "node:url";

import {
  ListSchemaMappingsCommand,
  EntityResolutionClient,
} from "@aws-sdk/client-entityresolution";
import data from "../inputs.json" with { type: "json" };

const region = "eu-west-1";
const erClient = new EntityResolutionClient({ region: region });

export const main = async () => {
  async function getInfo() {
    const listSchemaMappingsParams = {
      workflowName: `${data.inputs.workflowName}`,
      jobId: `${data.inputs.jobId}`,
    };
    try {
      const command = new ListSchemaMappingsCommand(listSchemaMappingsParams);
      const response = await erClient.send(command);
      const noOfSchemas = response.schemaList.length;
      for (let i = 0; i < noOfSchemas; i++) {
        console.log(
          `Schema Mapping Name: ${response.schemaList[i].schemaName} `,
        );
      }
    }
  }
}
```

```
    );  
  }  
} catch (caught) {  
  console.error(caught.message);  
  throw caught;  
}  
}
```

- Einzelheiten zur API finden Sie [ListSchemaMappings](#) in der AWS SDK für JavaScript API-Referenz.

StartMatchingJob

Das folgende Codebeispiel zeigt die Verwendung `StartMatchingJob`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
//The default inputs for this demo are read from the ../inputs.json.  
  
import { fileURLToPath } from "node:url";  
import {  
  StartMatchingJobCommand,  
  EntityResolutionClient,  
} from "@aws-sdk/client-entityresolution";  
import data from "../inputs.json" with { type: "json" };  
  
const region = "eu-west-1";  
const erClient = new EntityResolutionClient({ region: region });  
  
export const main = async () => {  
  const matchingJobOfWorkflowParams = {  
    workflowName: `${data.inputs.workflowName}`,
```



```

};
try {
  const command = new StartMatchingJobCommand(matchingJobOfWorkflowParams);
  const response = await erClient.send(command);
  console.log(`Job ID: ${response.jobID} \n
The matching job was successfully started.`);
} catch (caught) {
  console.error(caught.message);
  throw caught;
}
};

```

- Einzelheiten zur API finden Sie [StartMatchingJob](#) in der AWS SDK für JavaScript API-Referenz.

TagResource

Das folgende Codebeispiel zeigt die Verwendung `TagResource`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```

//The default inputs for this demo are read from the ../inputs.json.

import { fileURLToPath } from "node:url";

import {
  TagResourceCommand,
  EntityResolutionClient,
} from "@aws-sdk/client-entityresolution";
import data from "../inputs.json" with { type: "json" };

const region = "eu-west-1";
const erClient = new EntityResolutionClient({ region: region });

```

```
export const main = async () => {
  const tagResourceCommandParams = {
    resourceArn: `${data.inputs.schemaArn}`,
    tags: {
      tag1: "tag1Value",
      tag2: "tag2Value",
    },
  };
  try {
    const command = new TagResourceCommand(tagResourceCommandParams);
    const response = await erClient.send(command);
    console.log("Successfully tagged the resource.");
  } catch (caught) {
    console.error(caught.message);
    throw caught;
  }
};
```

- Einzelheiten zur API finden Sie [TagResource](#) in der AWS SDK für JavaScript API-Referenz.

EventBridge Beispiele für die Verwendung von SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie AWS SDK für JavaScript (v3) mit verwenden EventBridge.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien anzeigen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb eines Service aufrufen oder mit anderen AWS-Services kombinieren.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kodex finden.

Themen

- [Aktionen](#)
- [Szenarien](#)

Aktionen

PutEvents

Das folgende Codebeispiel zeigt die Verwendung von `PutEvents`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
import {
  EventBridgeClient,
  PutEventsCommand,
} from "@aws-sdk/client-eventbridge";

export const putEvents = async (
  source = "eventbridge.integration.test",
  detailType = "greeting",
  resources = [],
) => {
  const client = new EventBridgeClient({});

  const response = await client.send(
    new PutEventsCommand({
      Entries: [
        {
          Detail: JSON.stringify({ greeting: "Hello there." }),
          DetailType: detailType,
          Resources: resources,
          Source: source,
        },
      ],
    })
  );
}
```

```
    }},  
  );  
  
  console.log("PutEvents response:");  
  console.log(response);  
  // PutEvents response:  
  // {  
  //   '$metadata': {  
  //     httpStatusCode: 200,  
  //     requestId: '3d0df73d-dcea-4a23-ae0d-f5556a3ac109',  
  //     extendedRequestId: undefined,  
  //     cfId: undefined,  
  //     attempts: 1,  
  //     totalRetryDelay: 0  
  //   },  
  //   Entries: [ { EventId: '51620841-5af4-6402-d9bc-b77734991eb5' } ],  
  //   FailedEntryCount: 0  
  // }  
  
  return response;  
};
```

- Einzelheiten zur API finden Sie [PutEvents](#) in der AWS SDK für JavaScript API-Referenz.

PutRule

Das folgende Codebeispiel zeigt die Verwendung `PutRule`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
import { EventBridgeClient, PutRuleCommand } from "@aws-sdk/client-eventbridge";
```

```

export const putRule = async (
  ruleName = "some-rule",
  source = "some-source",
) => {
  const client = new EventBridgeClient({});

  const response = await client.send(
    new PutRuleCommand({
      Name: ruleName,
      EventPattern: JSON.stringify({ source: [source] }),
      State: "ENABLED",
      EventBusName: "default",
    }),
  );

  console.log("PutRule response:");
  console.log(response);
  // PutRule response:
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'd7292ced-1544-421b-842f-596326bc7072',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   RuleArn: 'arn:aws:events:us-east-1:xxxxxxxxxxxx:rule/
  EventBridgeTestRule-1696280037720'
  // }
  return response;
};

```

- Einzelheiten zur API finden Sie [PutRule](#) in der AWS SDK für JavaScript API-Referenz.

PutTargets

Das folgende Codebeispiel zeigt die Verwendung `PutTargets`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
import {
  EventBridgeClient,
  PutTargetsCommand,
} from "@aws-sdk/client-eventbridge";

export const putTarget = async (
  existingRuleName = "some-rule",
  targetArn = "arn:aws:lambda:us-east-1:000000000000:function:test-func",
  uniqueId = Date.now().toString(),
) => {
  const client = new EventBridgeClient({});
  const response = await client.send(
    new PutTargetsCommand({
      Rule: existingRuleName,
      Targets: [
        {
          Arn: targetArn,
          Id: uniqueId,
        },
      ],
    }),
  );

  console.log("PutTargets response:");
  console.log(response);
  // PutTargets response:
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'f5b23b9a-2c17-45c1-ad5c-f926c3692e3d',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
```

```
//      totalRetryDelay: 0
//    },
//    FailedEntries: [],
//    FailedEntryCount: 0
//  }

  return response;
};
```

- Einzelheiten zur API finden Sie [PutTargets](#) in der AWS SDK für JavaScript API-Referenz.

Szenarien

Verwendung geplanter Ereignisse zum Aufrufen einer Lambda-Funktion

Das folgende Codebeispiel zeigt, wie eine AWS Lambda Funktion erstellt wird, die durch ein von Amazon EventBridge geplantes Ereignis aufgerufen wird.

SDK für JavaScript (v3)

Zeigt, wie ein von Amazon EventBridge geplantes Ereignis erstellt wird, das eine AWS Lambda Funktion aufruft. Konfigurieren Sie so EventBridge, dass ein Cron-Ausdruck verwendet wird, um zu planen, wann die Lambda-Funktion aufgerufen wird. In diesem Beispiel erstellen Sie eine Lambda-Funktion mithilfe der JavaScript Lambda-Laufzeit-API. In diesem Beispiel werden verschiedene AWS Dienste aufgerufen, um einen bestimmten Anwendungsfall auszuführen. Dieses Beispiel zeigt, wie man eine App erstellt, die eine mobile Textnachricht an Ihre Mitarbeiter sendet, um ihnen zum einjährigen Jubiläum zu gratulieren.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

Dieses Beispiel ist auch verfügbar im [AWS SDK für JavaScript Entwicklerhandbuch für v3](#).

In diesem Beispiel verwendete Dienste

- CloudWatch Logs
- DynamoDB
- EventBridge
- Lambda

- Amazon SNS

Amazon Glacier-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK für JavaScript (v3) mit Amazon Glacier Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kodex finden.

Themen

- [Aktionen](#)

Aktionen

CreateVault

Das folgende Codebeispiel zeigt die Verwendung von `CreateVault`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie den Client.

```
const { GlacierClient } = require("@aws-sdk/client-glacier");
// Set the AWS Region.
const REGION = "REGION";
//Set the Redshift Service Object
const glacierClient = new GlacierClient({ region: REGION });
```



```
export { glacierClient };
```

Erstellen des Tresors.

```
// Load the SDK for JavaScript
import { CreateVaultCommand } from "@aws-sdk/client-glacier";
import { glacierClient } from "../libs/glacierClient.js";

// Set the parameters
const vaultname = "VAULT_NAME"; // VAULT_NAME
const params = { vaultName: vaultname };

const run = async () => {
  try {
    const data = await glacierClient.send(new CreateVaultCommand(params));
    console.log("Success, vault created!");
    return data; // For unit tests.
  } catch (err) {
    console.log("Error");
  }
};
run();
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [CreateVault](#) in der AWS SDK für JavaScript API-Referenz.

UploadArchive

Das folgende Codebeispiel zeigt die Verwendung `UploadArchive`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie den Client.

```
const { GlacierClient } = require("@aws-sdk/client-glacier");
// Set the AWS Region.
const REGION = "REGION";
//Set the Redshift Service Object
const glacierClient = new GlacierClient({ region: REGION });
export { glacierClient };
```

Hochladen des Archivs.

```
// Load the SDK for JavaScript
import { UploadArchiveCommand } from "@aws-sdk/client-glacier";
import { glacierClient } from "../libs/glacierClient.js";

// Set the parameters
const vaultname = "VAULT_NAME"; // VAULT_NAME

// Create a new service object and buffer
const buffer = new Buffer.alloc(2.5 * 1024 * 1024); // 2.5MB buffer
const params = { vaultName: vaultname, body: buffer };

const run = async () => {
  try {
    const data = await glacierClient.send(new UploadArchiveCommand(params));
    console.log("Archive ID", data.archiveId);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error uploading archive!", err);
  }
};
run();
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [UploadArchive](#) in der AWS SDK für JavaScript API-Referenz.

AWS Glue Beispiele für die Verwendung von SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie AWS SDK für JavaScript (v3) mit verwenden AWS Glue.

Bei Grundlagen handelt es sich um Codebeispiele, die Ihnen zeigen, wie Sie die wesentlichen Vorgänge innerhalb eines Services ausführen.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kodex finden.

Themen

- [Erste Schritte](#)
- [Grundlagen](#)
- [Aktionen](#)

Erste Schritte

Hallo AWS Glue

Das folgende Codebeispiel veranschaulicht, wie Sie mit der Verwendung von AWS Glue beginnen.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { ListJobsCommand, GlueClient } from "@aws-sdk/client-glue";
```

```
const client = new GlueClient({});

export const main = async () => {
  const command = new ListJobsCommand({});

  const { JobNames } = await client.send(command);
  const formattedJobNames = JobNames.join("\n");
  console.log("Job names: ");
  console.log(formattedJobNames);
  return JobNames;
};
```

- Einzelheiten zur API finden Sie [ListJobs](#) in der AWS SDK für JavaScript API-Referenz.

Grundlagen

Kennenlernen der Grundlagen

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Erstellen Sie einen Crawler, der einen öffentlichen Amazon-S3-Bucket crawlt und eine Datenbank mit CSV-formatierten Metadaten generiert.
- Führen Sie Informationen zu Datenbanken und Tabellen in Ihrem auf AWS Glue Data Catalog.
- Erstellen Sie einen Auftrag, um CSV-Daten aus dem S3-Bucket zu extrahieren, die Daten umzuwandeln und die JSON-formatierte Ausgabe in einen anderen S3-Bucket zu laden.
- Listen Sie Informationen zu Auftragsausführungen auf, zeigen Sie transformierte Daten an und bereinigen Sie Ressourcen.

Weitere Informationen finden Sie unter [Tutorial: Erste Schritte mit AWS Glue Studio](#).

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie einen Crawler und führen Sie ihn aus, der einen öffentlichen Amazon Simple Storage Service (Amazon S3)-Bucket crawlt und eine Metadatendatenbank generiert, die die gefundenen CSV-formatierten Daten beschreibt.

```
const createCrawler = (name, role, dbName, tablePrefix, s3TargetPath) => {
  const client = new GlueClient({});

  const command = new CreateCrawlerCommand({
    Name: name,
    Role: role,
    DatabaseName: dbName,
    TablePrefix: tablePrefix,
    Targets: {
      S3Targets: [{ Path: s3TargetPath }],
    },
  });

  return client.send(command);
};

const getCrawler = (name) => {
  const client = new GlueClient({});

  const command = new GetCrawlerCommand({
    Name: name,
  });

  return client.send(command);
};

const startCrawler = (name) => {
  const client = new GlueClient({});

  const command = new StartCrawlerCommand({
    Name: name,
  });

  return client.send(command);
};

const crawlerExists = async ({ getCrawler }, crawlerName) => {
  try {
    await getCrawler(crawlerName);
  }
```

```

        return true;
    } catch {
        return false;
    }
};

/**
 * @param {{ createCrawler: import('../.../actions/create-crawler.js').createCrawler }} actions
 */
const makeCreateCrawlerStep = (actions) => async (context) => {
    if (await crawlerExists(actions, process.env.CRAWLER_NAME)) {
        log("Crawler already exists. Skipping creation.");
    } else {
        await actions.createCrawler(
            process.env.CRAWLER_NAME,
            process.env.ROLE_NAME,
            process.env.DATABASE_NAME,
            process.env.TABLE_PREFIX,
            process.env.S3_TARGET_PATH,
        );

        log("Crawler created successfully.", { type: "success" });
    }

    return { ...context };
};

/**
 * @param {(name: string) => Promise<import('@aws-sdk/client-glue').GetCrawlerCommandOutput>} getCrawler
 * @param {string} crawlerName
 */
const waitForCrawler = async (getCrawler, crawlerName) => {
    const waitTimeInSeconds = 30;
    const { Crawler } = await getCrawler(crawlerName);

    if (!Crawler) {
        throw new Error(`Crawler with name ${crawlerName} not found.`);
    }

    if (Crawler.State === "READY") {
        return;
    }
};

```

```
log(`Crawler is ${Crawler.State}. Waiting ${waitTimeInSeconds} seconds...`);
await wait(waitTimeInSeconds);
return waitForCrawler(getCrawler, crawlerName);
};

const makeStartCrawlerStep =
  ({ startCrawler, getCrawler }) =>
  async (context) => {
    log("Starting crawler.");
    await startCrawler(process.env.CRAWLER_NAME);
    log("Crawler started.", { type: "success" });

    log("Waiting for crawler to finish running. This can take a while.");
    await waitForCrawler(getCrawler, process.env.CRAWLER_NAME);
    log("Crawler ready.", { type: "success" });

    return { ...context };
  };
};
```

Listen Sie Informationen zu Datenbanken und Tabellen in Ihrem auf AWS Glue Data Catalog.

```
const getDatabase = (name) => {
  const client = new GlueClient({});

  const command = new GetDatabaseCommand({
    Name: name,
  });

  return client.send(command);
};

const getTables = (databaseName) => {
  const client = new GlueClient({});

  const command = new GetTablesCommand({
    DatabaseName: databaseName,
  });

  return client.send(command);
};
```

```

const makeGetDatabaseStep =
  ({ getDatabase }) =>
  async (context) => {
    const {
      Database: { Name },
    } = await getDatabase(process.env.DATABASE_NAME);
    log(`Database: ${Name}`);
    return { ...context };
  };

/**
 * @param {{ getTables: () => Promise<import('@aws-sdk/client-glue').GetTablesCommandOutput> }} config
 */
const makeGetTablesStep =
  ({ getTables }) =>
  async (context) => {
    const { TableList } = await getTables(process.env.DATABASE_NAME);
    log("Tables:");
    log(TableList.map((table) => `  • ${table.Name}\n`));
    return { ...context };
  };

```

Erstellen und führen Sie einen Auftrag aus, der CSV-Daten aus dem Amazon-S3-Quell-Bucket extrahiert, sie durch Entfernen und Umbenennen von Feldern transformiert und die JSON-formatierte Ausgabe in einen anderen Amazon-S3-Bucket lädt.

```

const createJob = (name, role, scriptBucketName, scriptKey) => {
  const client = new GlueClient({});

  const command = new CreateJobCommand({
    Name: name,
    Role: role,
    Command: {
      Name: "glueetl",
      PythonVersion: "3",
      ScriptLocation: `s3://${scriptBucketName}/${scriptKey}`,
    },
    GlueVersion: "3.0",
  });

  return client.send(command);
}

```



```

};

const startJobRun = (jobName, dbName, tableName, bucketName) => {
  const client = new GlueClient({});

  const command = new StartJobRunCommand({
    JobName: jobName,
    Arguments: {
      "--input_database": dbName,
      "--input_table": tableName,
      "--output_bucket_url": `s3://${bucketName}/`,
    },
  });

  return client.send(command);
};

const makeCreateJobStep =
  ({ createJob }) =>
  async (context) => {
    log("Creating Job.");
    await createJob(
      process.env.JOB_NAME,
      process.env.ROLE_NAME,
      process.env.BUCKET_NAME,
      process.env.PYTHON_SCRIPT_KEY,
    );
    log("Job created.", { type: "success" });

    return { ...context };
  };

/**
 * @param {(name: string, runId: string) => Promise<import('@aws-sdk/client-glue').GetJobRunCommandOutput> } getJobRun
 * @param {string} jobName
 * @param {string} jobRunId
 */
const waitForJobRun = async (getJobRun, jobName, jobRunId) => {
  const waitTimeInSeconds = 30;
  const { JobRun } = await getJobRun(jobName, jobRunId);

  if (!JobRun) {
    throw new Error(`Job run with id ${jobRunId} not found.`);
  }
};

```

```

}

switch (JobRun.JobRunState) {
  case "FAILED":
  case "TIMEOUT":
  case "STOPPED":
  case "ERROR":
    throw new Error(
      `Job ${JobRun.JobRunState}. Error: ${JobRun.ErrorMessage}`,
    );
  case "SUCCEEDED":
    return;
  default:
    break;
}

log(
  `Job ${JobRun.JobRunState}. Waiting ${waitTimeInSeconds} more seconds...`,
);
await wait(waitTimeInSeconds);
return waitForJobRun(getJobRun, jobName, jobRunId);
};

/**
 * @param {{ prompter: { prompt: () => Promise<{ shouldOpen: boolean }>} }} context
 */
const promptToOpen = async (context) => {
  const { shouldOpen } = await context.prompter.prompt({
    name: "shouldOpen",
    type: "confirm",
    message: "Open the output bucket in your browser?",
  });

  if (shouldOpen) {
    return open(
      `https://s3.console.aws.amazon.com/s3/buckets/${process.env.BUCKET_NAME} to
view the output.`,
    );
  }
};

const makeStartJobRunStep =
  ({ startJobRun, getJobRun }) =>
  async (context) => {

```

```

log("Starting job.");
const { JobRunId } = await startJobRun(
  process.env.JOB_NAME,
  process.env.DATABASE_NAME,
  process.env.TABLE_NAME,
  process.env.BUCKET_NAME,
);
log("Job started.", { type: "success" });

log("Waiting for job to finish running. This can take a while.");
await waitForJobRun(getJobRun, process.env.JOB_NAME, JobRunId);
log("Job run succeeded.", { type: "success" });

await promptToOpen(context);

return { ...context };
};

```

Listet Informationen über Auftragsausführungen auf und zeigt einige der transformierten Daten an.

```

const getJobRuns = (jobName) => {
  const client = new GlueClient({});
  const command = new GetJobRunsCommand({
    JobName: jobName,
  });

  return client.send(command);
};

const getJobRun = (jobName, jobRunId) => {
  const client = new GlueClient({});
  const command = new GetJobRunCommand({
    JobName: jobName,
    RunId: jobRunId,
  });

  return client.send(command);
};

/**
 * @typedef {{ prompter: { prompt: () => Promise<{jobName: string}> } }} Context
 */

```

```

/**
 * @typedef {() => Promise<import('@aws-sdk/client-glue').GetJobRunCommandOutput>}
getJobRun
 */

/**
 * @typedef {() => Promise<import('@aws-sdk/client-glue').GetJobRunsCommandOutput>}
getJobRuns
 */

/**
 *
 * @param {getJobRun} getJobRun
 * @param {string} jobName
 * @param {string} jobRunId
 */
const logJobRunDetails = async (getJobRun, jobName, jobRunId) => {
  const { JobRun } = await getJobRun(jobName, jobRunId);
  log(JobRun, { type: "object" });
};

/**
 *
 * @param {{getJobRuns: getJobRuns, getJobRun: getJobRun }} funcs
 */
const makePickJobRunStep =
  ({ getJobRuns, getJobRun }) =>
  async (/** @type { Context } */ context) => {
    if (context.selectedJobName) {
      const { JobRuns } = await getJobRuns(context.selectedJobName);

      const { jobRunId } = await context.prompter.prompt({
        name: "jobRunId",
        type: "list",
        message: "Select a job run to see details.",
        choices: JobRuns.map((run) => run.Id),
      });

      logJobRunDetails(getJobRun, context.selectedJobName, jobRunId);
    }

    return { ...context };
  };

```

Löscht alle Ressourcen, die von der Demo erstellt wurden.

```
const deleteJob = (jobName) => {
  const client = new GlueClient({});

  const command = new DeleteJobCommand({
    JobName: jobName,
  });

  return client.send(command);
};

const deleteTable = (databaseName, tableName) => {
  const client = new GlueClient({});

  const command = new DeleteTableCommand({
    DatabaseName: databaseName,
    Name: tableName,
  });

  return client.send(command);
};

const deleteDatabase = (databaseName) => {
  const client = new GlueClient({});

  const command = new DeleteDatabaseCommand({
    Name: databaseName,
  });

  return client.send(command);
};

const deleteCrawler = (crawlerName) => {
  const client = new GlueClient({});

  const command = new DeleteCrawlerCommand({
    Name: crawlerName,
  });

  return client.send(command);
};
```

```

};

/**
 *
 * @param {import('.././../actions/delete-job.js').deleteJob} deleteJobFn
 * @param {string[]} jobNames
 * @param {{ prompter: { prompt: () => Promise<any> }}} context
 */
const handleDeleteJobs = async (deleteJobFn, jobNames, context) => {
  /**
   * @type {{ selectedJobNames: string[] }}
   */
  const { selectedJobNames } = await context.prompter.prompt({
    name: "selectedJobNames",
    type: "checkbox",
    message: "Let's clean up jobs. Select jobs to delete.",
    choices: jobNames,
  });

  if (selectedJobNames.length === 0) {
    log("No jobs selected.");
  } else {
    log("Deleting jobs.");
    await Promise.all(
      selectedJobNames.map((n) => deleteJobFn(n).catch(console.error)),
    );
    log("Jobs deleted.", { type: "success" });
  }
};

/**
 * @param {{
 *   listJobs: import('.././../actions/list-jobs.js').listJobs,
 *   deleteJob: import('.././../actions/delete-job.js').deleteJob
 * }} config
 */
const makeCleanUpJobsStep =
  ({ listJobs, deleteJob }) =>
  async (context) => {
    const { JobNames } = await listJobs();
    if (JobNames.length > 0) {
      await handleDeleteJobs(deleteJob, JobNames, context);
    }
  }

```

```

    return { ...context };
  };

/**
 * @param {import('.././../actions/delete-table.js').deleteTable} deleteTable
 * @param {string} databaseName
 * @param {string[]} tableNames
 */
const deleteTables = (deleteTable, databaseName, tableNames) =>
  Promise.all(
    tableNames.map((tableName) =>
      deleteTable(databaseName, tableName).catch(console.error),
    ),
  );

/**
 * @param {{
 *   getTables: import('.././../actions/get-tables.js').getTables,
 *   deleteTable: import('.././../actions/delete-table.js').deleteTable
 * }} config
 */
const makeCleanUpTablesStep =
  ({ getTables, deleteTable }) =>
  /**
   * @param {{ prompter: { prompt: () => Promise<any> }}} context
   */
  async (context) => {
    const { TableList } = await getTables(process.env.DATABASE_NAME).catch(
      () => ({ TableList: null }),
    );

    if (TableList && TableList.length > 0) {
      /**
       * @type {{ tableNames: string[] }}
       */
      const { tableNames } = await context.prompter.prompt({
        name: "tableNames",
        type: "checkbox",
        message: "Let's clean up tables. Select tables to delete.",
        choices: TableList.map((t) => t.Name),
      });

      if (tableNames.length === 0) {
        log("No tables selected.");
      }
    }
  }

```

```

    } else {
      log("Deleting tables.");
      await deleteTables(deleteTable, process.env.DATABASE_NAME, tableNames);
      log("Tables deleted.", { type: "success" });
    }
  }

  return { ...context };
};

/**
 * @param {import('../../actions/delete-database.js').deleteDatabase}
deleteDatabase
 * @param {string[]} databaseNames
 */
const deleteDatabases = (deleteDatabase, databaseNames) =>
  Promise.all(
    databaseNames.map((dbName) => deleteDatabase(dbName).catch(console.error)),
  );

/**
 * @param {{
 *   getDatabases: import('../../actions/get-databases.js').getDatabases
 *   deleteDatabase: import('../../actions/delete-database.js').deleteDatabase
 * }} config
 */
const makeCleanUpDatabasesStep =
  ({ getDatabases, deleteDatabase }) =>
  /**
   * @param {{ prompter: { prompt: () => Promise<any> } } } context
   */
  async (context) => {
    const { DatabaseList } = await getDatabases();

    if (DatabaseList.length > 0) {
      /** @type {{ dbName: string[] }} */
      const { dbName } = await context.prompter.prompt({
        name: "dbNames",
        type: "checkbox",
        message: "Let's clean up databases. Select databases to delete.",
        choices: DatabaseList.map((db) => db.Name),
      });

      if (dbName.length === 0) {

```



```
        log("No databases selected.");
    } else {
        log("Deleting databases.");
        await deleteDatabases(deleteDatabase, dbNames);
        log("Databases deleted.", { type: "success" });
    }
}

return { ...context };
};

const cleanUpCrawlerStep = async (context) => {
    log("Deleting crawler.");

    try {
        await deleteCrawler(process.env.CRAWLER_NAME);
        log("Crawler deleted.", { type: "success" });
    } catch (err) {
        if (err.name === "EntityNotFoundException") {
            log("Crawler is already deleted.");
        } else {
            throw err;
        }
    }

    return { ...context };
};
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für JavaScript - API-Referenz.

- [CreateCrawler](#)
- [CreateJob](#)
- [DeleteCrawler](#)
- [DeleteDatabase](#)
- [DeleteJob](#)
- [DeleteTable](#)
- [GetCrawler](#)
- [GetDatabase](#)

- [GetDatabases](#)
- [GetJob](#)
- [GetJobRun](#)
- [GetJobRuns](#)
- [GetTables](#)
- [ListJobs](#)
- [StartCrawler](#)
- [StartJobRun](#)

Aktionen

CreateCrawler

Das folgende Codebeispiel zeigt die Verwendung `CreateCrawler`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
const createCrawler = (name, role, dbName, tablePrefix, s3TargetPath) => {
  const client = new GlueClient({});

  const command = new CreateCrawlerCommand({
    Name: name,
    Role: role,
    DatabaseName: dbName,
    TablePrefix: tablePrefix,
    Targets: {
      S3Targets: [{ Path: s3TargetPath }],
    },
  });

  return client.send(command);
};
```

- Einzelheiten zur API finden Sie [CreateCrawler](#) in der AWS SDK für JavaScript API-Referenz.

CreateJob

Das folgende Codebeispiel zeigt die Verwendung `CreateJob`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
const createJob = (name, role, scriptBucketName, scriptKey) => {
  const client = new GlueClient({});

  const command = new CreateJobCommand({
    Name: name,
    Role: role,
    Command: {
      Name: "glueetl",
      PythonVersion: "3",
      ScriptLocation: `s3://${scriptBucketName}/${scriptKey}`,
    },
    GlueVersion: "3.0",
  });

  return client.send(command);
};
```

- Einzelheiten zur API finden Sie [CreateJob](#) in der AWS SDK für JavaScript API-Referenz.

DeleteCrawler

Das folgende Codebeispiel zeigt die Verwendung `DeleteCrawler`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
const deleteCrawler = (crawlerName) => {  
  const client = new GlueClient({});  
  
  const command = new DeleteCrawlerCommand({  
    Name: crawlerName,  
  });  
  
  return client.send(command);  
};
```

- Einzelheiten zur API finden Sie [DeleteCrawler](#) in der AWS SDK für JavaScript API-Referenz.

DeleteDatabase

Das folgende Codebeispiel zeigt die Verwendung `DeleteDatabase`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
const deleteDatabase = (databaseName) => {  
  const client = new GlueClient({});  
  
  const command = new DeleteDatabaseCommand({  
    Name: databaseName,  
  });  
};
```

```
    return client.send(command);  
};
```

- Einzelheiten zur API finden Sie [DeleteDatabase](#) in der AWS SDK für JavaScript API-Referenz.

DeleteJob

Das folgende Codebeispiel zeigt die Verwendung `DeleteJob`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
const deleteJob = (jobName) => {  
    const client = new GlueClient({});  
  
    const command = new DeleteJobCommand({  
        JobName: jobName,  
    });  
  
    return client.send(command);  
};
```

- Einzelheiten zur API finden Sie [DeleteJob](#) in der AWS SDK für JavaScript API-Referenz.

DeleteTable

Das folgende Codebeispiel zeigt die Verwendung `DeleteTable`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
const deleteTable = (databaseName, tableName) => {  
  const client = new GlueClient({});  
  
  const command = new DeleteTableCommand({  
    DatabaseName: databaseName,  
    Name: tableName,  
  });  
  
  return client.send(command);  
};
```

- Einzelheiten zur API finden Sie [DeleteTable](#) in der AWS SDK für JavaScript API-Referenz.

GetCrawler

Das folgende Codebeispiel zeigt die Verwendung `GetCrawler`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
const getCrawler = (name) => {  
  const client = new GlueClient({});  
  
  const command = new GetCrawlerCommand({  
    Name: name,  
  });  
  
  return client.send(command);  
};
```

```
});  
  
    return client.send(command);  
};
```

- Einzelheiten zur API finden Sie [GetCrawler](#) in der AWS SDK für JavaScript API-Referenz.

GetDatabase

Das folgende Codebeispiel zeigt die Verwendung `GetDatabase`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
const getDatabase = (name) => {  
    const client = new GlueClient({});  
  
    const command = new GetDatabaseCommand({  
        Name: name,  
    });  
  
    return client.send(command);  
};
```

- Einzelheiten zur API finden Sie [GetDatabase](#) in der AWS SDK für JavaScript API-Referenz.

GetDatabases

Das folgende Codebeispiel zeigt die Verwendung `GetDatabases`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
const getDatabases = () => {  
  const client = new GlueClient({});  
  
  const command = new GetDatabasesCommand({});  
  
  return client.send(command);  
};
```

- Einzelheiten zur API finden Sie [GetDatabases](#) in der AWS SDK für JavaScript API-Referenz.

GetJob

Das folgende Codebeispiel zeigt die Verwendung `GetJob`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
const getJob = (jobName) => {  
  const client = new GlueClient({});  
  
  const command = new GetJobCommand({  
    JobName: jobName,  
  });  
  
  return client.send(command);  
};
```


- Einzelheiten zur API finden Sie [GetJob](#) in der AWS SDK für JavaScript API-Referenz.

GetJobRun

Das folgende Codebeispiel zeigt die Verwendung `GetJobRun`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
const getJobRun = (jobName, jobRunId) => {  
  const client = new GlueClient({});  
  const command = new GetJobRunCommand({  
    JobName: jobName,  
    RunId: jobRunId,  
  });  
  
  return client.send(command);  
};
```

- Einzelheiten zur API finden Sie [GetJobRun](#) in der AWS SDK für JavaScript API-Referenz.

GetJobRuns

Das folgende Codebeispiel zeigt die Verwendung `GetJobRuns`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
const getJobRuns = (jobName) => {  
  const client = new GlueClient({});  
  const command = new GetJobRunsCommand({  
    JobName: jobName,  
  });  
  
  return client.send(command);  
};
```

- Einzelheiten zur API finden Sie [GetJobRuns](#) in der AWS SDK für JavaScript API-Referenz.

GetTables

Das folgende Codebeispiel zeigt die Verwendung `GetTables`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
const getTables = (databaseName) => {  
  const client = new GlueClient({});  
  
  const command = new GetTablesCommand({  
    DatabaseName: databaseName,  
  });  
  
  return client.send(command);  
};
```

- Einzelheiten zur API finden Sie [GetTables](#) in der AWS SDK für JavaScript API-Referenz.

ListJobs

Das folgende Codebeispiel zeigt die Verwendung `ListJobs`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
const listJobs = () => {  
  const client = new GlueClient({});  
  
  const command = new ListJobsCommand({});  
  
  return client.send(command);  
};
```

- Einzelheiten zur API finden Sie [ListJobs](#) in der AWS SDK für JavaScript API-Referenz.

StartCrawler

Das folgende Codebeispiel zeigt die Verwendung `StartCrawler`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
const startCrawler = (name) => {  
  const client = new GlueClient({});  
  
  const command = new StartCrawlerCommand({  
    Name: name,  
  });  
  
  return client.send(command);  
};
```

- Einzelheiten zur API finden Sie [StartCrawler](#) in der AWS SDK für JavaScript API-Referenz.

StartJobRun

Das folgende Codebeispiel zeigt die Verwendung `StartJobRun`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
const startJobRun = (jobName, dbName, tableName, bucketName) => {
  const client = new GlueClient({});

  const command = new StartJobRunCommand({
    JobName: jobName,
    Arguments: {
      "--input_database": dbName,
      "--input_table": tableName,
      "--output_bucket_url": `s3://${bucketName}/`,
    },
  });

  return client.send(command);
};
```

- Einzelheiten zur API finden Sie [StartJobRun](#) in der AWS SDK für JavaScript API-Referenz.

HealthImaging Beispiele für die Verwendung von SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie AWS SDK für JavaScript (v3) mit verwenden HealthImaging.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien anzeigen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb eines Service aufrufen oder mit anderen AWS-Services kombinieren.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kodex finden.

Themen

- [Erste Schritte](#)
- [Aktionen](#)
- [Szenarien](#)

Erste Schritte

Hallo HealthImaging

Das folgende Codebeispiel veranschaulicht, wie Sie mit der Verwendung von HealthImaging beginnen.

SDK für JavaScript (v3)

```
import {
  ListDatastoresCommand,
  MedicalImagingClient,
} from "@aws-sdk/client-medical-imaging";

// When no region or credentials are provided, the SDK will use the
// region and credentials from the local AWS config.
const client = new MedicalImagingClient({});

export const helloMedicalImaging = async () => {
  const command = new ListDatastoresCommand({});

  const { datastoreSummaries } = await client.send(command);
  console.log("Datastores: ");
  console.log(datastoreSummaries.map((item) => item.datastoreName).join("\n"));
  return datastoreSummaries;
};
```

- Einzelheiten zur API finden Sie [ListDatastores](#) in der AWS SDK für JavaScript API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Aktionen

CopyImageSet

Das folgende Codebeispiel zeigt die Verwendung `CopyImageSet`.

SDK für JavaScript (v3)

Hilfsfunktion zum Kopieren eines Imagesatzes.

```
import { CopyImageSetCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imageSetId - The source image set ID.
 * @param {string} sourceVersionId - The source version ID.
 * @param {string} destinationImageSetId - The optional ID of the destination image
 * set.
 * @param {string} destinationVersionId - The optional version ID of the destination
 * image set.
 * @param {boolean} force - Force the copy action.
 * @param {[string]} copySubsets - A subset of instance IDs to copy.
 */
export const copyImageSet = async (
  datastoreId = "xxxxxxxxxx",
  imageSetId = "xxxxxxxxxx",
  sourceVersionId = "1",
  destinationImageSetId = "",
  destinationVersionId = "",
  force = false,
  copySubsets = [],
) => {
```

```
try {
  const params = {
    datastoreId: datastoreId,
    sourceImageSetId: imageSetId,
    copyImageSetInformation: {
      sourceImageSet: { latestVersionId: sourceVersionId },
    },
    force: force,
  };
  if (destinationImageSetId !== "" && destinationVersionId !== "") {
    params.copyImageSetInformation.destinationImageSet = {
      imageSetId: destinationImageSetId,
      latestVersionId: destinationVersionId,
    };
  }

  if (copySubsets.length > 0) {
    let copySubsetsJson;
    copySubsetsJson = {
      SchemaVersion: 1.1,
      Study: {
        Series: {
          imageSetId: {
            Instances: {},
          },
        },
      },
    };

    for (let i = 0; i < copySubsets.length; i++) {
      copySubsetsJson.Study.Series.imageSetId.Instances[copySubsets[i]] = {};
    }

    params.copyImageSetInformation.dicomCopies = copySubsetsJson;
  }

  const response = await medicalImagingClient.send(
    new CopyImageSetCommand(params),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'd9b219ce-cc48-4a44-a5b2-c5c3068f1ee8',
```

```

//      extendedRequestId: undefined,
//      cfId: undefined,
//      attempts: 1,
//      totalRetryDelay: 0
//    },
//    datastoreId: 'xxxxxxxxxxxxxx',
//    destinationImageSetProperties: {
//      createdAt: 2023-09-27T19:46:21.824Z,
//      imageSetArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxxxxx:datastore/xxxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxxxxxxxx',
//      imageSetId: 'xxxxxxxxxxxxxxxxxx',
//      imageSetState: 'LOCKED',
//      imageSetWorkflowStatus: 'COPYING',
//      latestVersionId: '1',
//      updatedAt: 2023-09-27T19:46:21.824Z
//    },
//    sourceImageSetProperties: {
//      createdAt: 2023-09-22T14:49:26.427Z,
//      imageSetArn: 'arn:aws:medical-imaging:us-
east-1:xxxxxxxxxxxx:datastore/xxxxxxxxxxxxx/imageset/xxxxxxxxxxxxxxxxxxxxxx',
//      imageSetId: 'xxxxxxxxxxxxxxxxxx',
//      imageSetState: 'LOCKED',
//      imageSetWorkflowStatus: 'COPYING_WITH_READ_ONLY_ACCESS',
//      latestVersionId: '4',
//      updatedAt: 2023-09-27T19:46:21.824Z
//    }
//  }
return response;
} catch (err) {
  console.error(err);
}
};

```

Kopieren Sie einen Imagesatz ohne Ziel.

```

await copyImageSet(
  "12345678901234567890123456789012",
  "12345678901234567890123456789012",
  "1",
);

```


Kopieren Sie einen Imagesatz mit Ziel.

```
await copyImageSet(  
  "12345678901234567890123456789012",  
  "12345678901234567890123456789012",  
  "1",  
  "12345678901234567890123456789012",  
  "1",  
  false,  
);
```

Kopieren Sie eine Teilmenge eines Imagesatzes mit einem Ziel und erzwingen Sie den Kopiervorgang.

```
await copyImageSet(  
  "12345678901234567890123456789012",  
  "12345678901234567890123456789012",  
  "1",  
  "12345678901234567890123456789012",  
  "1",  
  true,  
  ["12345678901234567890123456789012", "11223344556677889900112233445566"],  
);
```

- Einzelheiten zur API finden Sie [CopyImageSet](#) in der AWS SDK für JavaScript API-Referenz.

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

CreateDatastore

Das folgende Codebeispiel zeigt die Verwendung `CreateDatastore`.

SDK für JavaScript (v3)

```
import { CreateDatastoreCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreName - The name of the data store to create.
 */
export const createDatastore = async (datastoreName = "DATASTORE_NAME") => {
  const response = await medicalImagingClient.send(
    new CreateDatastoreCommand({ datastoreName: datastoreName }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'a71cd65f-2382-49bf-b682-f9209d8d399b',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //   datastoreStatus: 'CREATING'
  // }
  return response;
};
```

- Einzelheiten zur API finden Sie [CreateDatastore](#) in der AWS SDK für JavaScript API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

DeleteDatastore

Das folgende Codebeispiel zeigt die Verwendung `DeleteDatastore`.

SDK für JavaScript (v3)

```
import { DeleteDatastoreCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store to delete.
 */
export const deleteDatastore = async (datastoreId = "DATASTORE_ID") => {
  const response = await medicalImagingClient.send(
    new DeleteDatastoreCommand({ datastoreId }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'f5beb409-678d-48c9-9173-9a001ee1ebb1',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //   datastoreStatus: 'DELETING'
  // }

  return response;
};
```

- Einzelheiten zur API finden Sie [DeleteDatastore](#) in der AWS SDK für JavaScript API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

DeleteImageSet

Das folgende Codebeispiel zeigt die Verwendung `DeleteImageSet`.

SDK für JavaScript (v3)

```

import { DeleteImageSetCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The data store ID.
 * @param {string} imageSetId - The image set ID.
 */
export const deleteImageSet = async (
  datastoreId = "xxxxxxxxxxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxxxxxxxxxx",
) => {
  const response = await medicalImagingClient.send(
    new DeleteImageSetCommand({
      datastoreId: datastoreId,
      imageSetId: imageSetId,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '6267bbd2-eaa5-4a50-8ee8-8fddf535cf73',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   datastoreId: 'xxxxxxxxxxxxxxxxxxxx',
  //   imageSetId: 'xxxxxxxxxxxxxxxxxxxx',
  //   imageSetState: 'LOCKED',
  //   imageSetWorkflowStatus: 'DELETING'
  // }
  return response;
};

```

- Einzelheiten zur API finden Sie [DeleteImageSet](#) in der AWS SDK für JavaScript API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

GetDICOMImportJob

Das folgende Codebeispiel zeigt die Verwendung `GetDICOMImportJob`.

SDK für JavaScript (v3)

```
import { GetDICOMImportJobCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} jobId - The ID of the import job.
 */
export const getDICOMImportJob = async (
  datastoreId = "xxxxxxxxxxxxxxxxxxxxxx",
  jobId = "xxxxxxxxxxxxxxxxxxxxxx",
) => {
  const response = await medicalImagingClient.send(
    new GetDICOMImportJobCommand({ datastoreId: datastoreId, jobId: jobId }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'a2637936-78ea-44e7-98b8-7a87d95dfaee',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   jobProperties: {
  //     dataAccessRoleArn: 'arn:aws:iam::xxxxxxxxxxxx:role/dicom_import',
  //     datastoreId: 'xxxxxxxxxxxxxxxxxxxxxx',
  //     endedAt: 2023-09-19T17:29:21.753Z,
  //     inputS3Uri: 's3://healthimaging-source/CTStudy/',
  //     jobId: 'xxxxxxxxxxxxxxxxxxxxxx',
  //     jobName: 'job_1',
  //   },
  // }
```

```
//          jobStatus: 'COMPLETED',
//          outputS3Uri: 's3://health-imaging-dest/
output_ct/'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'-DicomImport-'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'/',
//          submittedAt: 2023-09-19T17:27:25.143Z
//      }
// }

return response;
};
```

- Einzelheiten zur API finden Sie unter [Get DICOMImport Job](#) in der AWS SDK für JavaScript API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

GetDatastore

Das folgende Codebeispiel zeigt die Verwendung `GetDatastore`.

SDK für JavaScript (v3)

```
import { GetDatastoreCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreID - The ID of the data store.
 */
export const getDatastore = async (datastoreID = "DATASTORE_ID") => {
  const response = await medicalImagingClient.send(
    new GetDatastoreCommand({ datastoreId: datastoreID }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '55ea7d2e-222c-4a6a-871e-4f591f40cadb',
  //     extendedRequestId: undefined,
```

```
//      cfId: undefined,
//      attempts: 1,
//      totalRetryDelay: 0
//    },
//    datastoreProperties: {
//      createdAt: 2023-08-04T18:50:36.239Z,
//      datastoreArn: 'arn:aws:medical-imaging:us-east-1:xxxxxxxxx:datastore/
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//      datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//      datastoreName: 'my_datastore',
//      datastoreStatus: 'ACTIVE',
//      updatedAt: 2023-08-04T18:50:36.239Z
//    }
//  }
return response.datastoreProperties;
};
```

- Einzelheiten zur API finden Sie [GetDatastore](#) in der AWS SDK für JavaScript API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

GetImageFrame

Das folgende Codebeispiel zeigt die Verwendung `GetImageFrame`.

SDK für JavaScript (v3)

```
import { GetImageFrameCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} imageFrameFileName - The name of the file for the HTJ2K-encoded
 * image frame.
 * @param {string} datastoreID - The data store's ID.
 * @param {string} imageSetID - The image set's ID.
 * @param {string} imageFrameID - The image frame's ID.
 */
export const getImageFrame = async (
```

```

imageFrameFileName = "image.jph",
datastoreId = "DATASTORE_ID",
imageSetID = "IMAGE_SET_ID",
imageFrameID = "IMAGE_FRAME_ID",
) => {
  const response = await medicalImagingClient.send(
    new GetImageFrameCommand({
      datastoreId: datastoreID,
      imageSetId: imageSetID,
      imageFrameInformation: { imageFrameId: imageFrameID },
    }),
  );
  const buffer = await response.imageFrameBlob.transformToByteArray();
  writeFileSync(imageFrameFileName, buffer);

  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'e4ab42a5-25a3-4377-873f-374ecf4380e1',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   contentType: 'application/octet-stream',
  //   imageFrameBlob: <ref *1> IncomingMessage {}
  // }
  return response;
};

```

- Einzelheiten zur API finden Sie [GetImageFrame](#) in der AWS SDK für JavaScript API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

GetImageSet

Das folgende Codebeispiel zeigt die Verwendung `GetImageSet`.

SDK für JavaScript (v3)

```
import { GetImageSetCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imageSetId - The ID of the image set.
 * @param {string} imageSetVersion - The optional version of the image set.
 */
export const getImageSet = async (
  datastoreId = "xxxxxxxxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxxxxxxxx",
  imageSetVersion = "",
) => {
  const params = { datastoreId: datastoreId, imageSetId: imageSetId };
  if (imageSetVersion !== "") {
    params.imageSetVersion = imageSetVersion;
  }
  const response = await medicalImagingClient.send(
    new GetImageSetCommand(params),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '0615c161-410d-4d06-9d8c-6e1241bb0a5a',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   createdAt: 2023-09-22T14:49:26.427Z,
  //   datastoreId: 'xxxxxxxxxxxxxxxxxx',
  //   imageSetArn: 'arn:aws:medical-imaging:us-east-1:xxxxxxxxxx:datastore/xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx/imageSet/xxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //   imageSetId: 'xxxxxxxxxxxxxxxxxx',
  //   imageSetState: 'ACTIVE',
  //   imageSetWorkflowStatus: 'CREATED',
  //   updatedAt: 2023-09-22T14:49:26.427Z,
  //   versionId: '1'
  // }
```

```
    return response;
  };
```

- Einzelheiten zur API finden Sie [GetImageSet](#) in der AWS SDK für JavaScript API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

GetImageSetMetadata

Das folgende Codebeispiel zeigt die Verwendung `GetImageSetMetadata`.

SDK für JavaScript (v3)

Hilfsfunktion zum Abrufen von Bildsatz-Metadaten.

```
import { GetImageSetMetadataCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";
import { writeFileSync } from "node:fs";

/**
 * @param {string} metadataFileName - The name of the file for the gzipped metadata.
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imagesetId - The ID of the image set.
 * @param {string} versionID - The optional version ID of the image set.
 */
export const getImageSetMetadata = async (
  metadataFileName = "metadata.json.gzip",
  datastoreId = "xxxxxxxxxxxxxxxx",
  imagesetId = "xxxxxxxxxxxxxxxx",
  versionID = "",
) => {
  const params = { datastoreId: datastoreId, imageSetId: imagesetId };

  if (versionID) {
    params.versionID = versionID;
  }
}
```

```
const response = await medicalImagingClient.send(
  new GetImageSetMetadataCommand(params),
);
const buffer = await response.imageSetMetadataBlob.transformToByteArray();
writeFileSync(metadataFileName, buffer);

console.log(response);
// {
//   '$metadata': {
//     httpStatusCode: 200,
//     requestId: '5219b274-30ff-4986-8cab-48753de3a599',
//     extendedRequestId: undefined,
//     cfId: undefined,
//     attempts: 1,
//     totalRetryDelay: 0
//   },
//   contentType: 'application/json',
//   contentEncoding: 'gzip',
//   imageSetMetadataBlob: <ref *1> IncomingMessage {}
// }

return response;
};
```

Rufen Sie Bildsatz-Metadaten ohne Version ab.

```
try {
  await getImageSetMetadata(
    "metadata.json.gzip",
    "12345678901234567890123456789012",
    "12345678901234567890123456789012",
  );
} catch (err) {
  console.log("Error", err);
}
```

Rufen Sie Bildsatz-Metadaten mit Version ab.

```
try {
  await getImageSetMetadata(
```

```

    "metadata2.json.gzip",
    "12345678901234567890123456789012",
    "12345678901234567890123456789012",
    "1",
  );
} catch (err) {
  console.log("Error", err);
}

```

- Einzelheiten zur API finden Sie [GetImageSetMetadata](#) in der AWS SDK für JavaScript API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

ListDICOMImportJobs

Das folgende Codebeispiel zeigt die Verwendung `ListDICOMImportJobs`.

SDK für JavaScript (v3)

```

import { paginateListDICOMImportJobs } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 */
export const listDICOMImportJobs = async (
  datastoreId = "xxxxxxxxxxxxxxxxxxxx",
) => {
  const paginatorConfig = {
    client: medicalImagingClient,
    pageSize: 50,
  };

  const commandParams = { datastoreId: datastoreId };
  const paginator = paginateListDICOMImportJobs(paginatorConfig, commandParams);

```

```

const jobSummaries = [];
for await (const page of paginator) {
  // Each page contains a list of `jobSummaries`. The list is truncated if is
  // larger than `pageSize`.
  jobSummaries.push(...page.jobSummaries);
  console.log(page);
}
// {
//   '$metadata': {
//     httpStatusCode: 200,
//     requestId: '3c20c66e-0797-446a-a1d8-91b742fd15a0',
//     extendedRequestId: undefined,
//     cfId: undefined,
//     attempts: 1,
//     totalRetryDelay: 0
//   },
//   jobSummaries: [
//     {
//       dataAccessRoleArn: 'arn:aws:iam:xxxxxxxxxxxx:role/dicom_import',
//       datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//       endedAt: 2023-09-22T14:49:51.351Z,
//       jobId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxx',
//       jobName: 'test-1',
//       jobStatus: 'COMPLETED',
//       submittedAt: 2023-09-22T14:48:45.767Z
//     }
//   ]
// }

return jobSummaries;
};

```

- Einzelheiten zur API finden Sie unter [DICOMImportJobs auflisten](#) in der AWS SDK für JavaScript API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

ListDatastores

Das folgende Codebeispiel zeigt die Verwendung `ListDatastores`.

SDK für JavaScript (v3)

```
import { paginateListDatastores } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

export const listDatastores = async () => {
  const paginatorConfig = {
    client: medicalImagingClient,
    pageSize: 50,
  };

  const commandParams = {};
  const paginator = paginateListDatastores(paginatorConfig, commandParams);

  /**
   * @type {import("@aws-sdk/client-medical-imaging").DatastoreSummary[]}
   */
  const datastoreSummaries = [];
  for await (const page of paginator) {
    // Each page contains a list of `jobSummaries`. The list is truncated if is
    // larger than `pageSize`.
    datastoreSummaries.push(...page.datastoreSummaries);
    console.log(page);
  }
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '6aa99231-d9c2-4716-a46e-edb830116fa3',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   datastoreSummaries: [
  //     {
  //       createdAt: 2023-08-04T18:49:54.429Z,
  //       datastoreArn: 'arn:aws:medical-imaging:us-east-1:xxxxxxxxx:datastore/
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //       datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
  //       datastoreName: 'my_datastore',

```

```
//      datastoreStatus: 'ACTIVE',
//      updatedAt: 2023-08-04T18:49:54.429Z
//    }
//    ...
//  ]
// }

return datastoreSummaries;
};
```

- Einzelheiten zur API finden Sie [ListDatastores](#) in der AWS SDK für JavaScript API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

ListImageSetVersions

Das folgende Codebeispiel zeigt die Verwendung `ListImageSetVersions`.

SDK für JavaScript (v3)

```
import { paginateListImageSetVersions } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} imageSetId - The ID of the image set.
 */
export const listImageSetVersions = async (
  datastoreId = "xxxxxxxxxxxx",
  imageSetId = "xxxxxxxxxxxx",
) => {
  const paginatorConfig = {
    client: medicalImagingClient,
    pageSize: 50,
  };

  const commandParams = { datastoreId, imageSetId };
```

```

const paginator = paginateListImageSetVersions(
  paginatorConfig,
  commandParams,
);

const imageSetPropertiesList = [];
for await (const page of paginator) {
  // Each page contains a list of `jobSummaries`. The list is truncated if is
  // larger than `pageSize`.
  imageSetPropertiesList.push(...page.imageSetPropertiesList);
  console.log(page);
}
// {
//   '$metadata': {
//     httpStatusCode: 200,
//     requestId: '74590b37-a002-4827-83f2-3c590279c742',
//     extendedRequestId: undefined,
//     cfId: undefined,
//     attempts: 1,
//     totalRetryDelay: 0
//   },
//   imageSetPropertiesList: [
//     {
//       ImageSetWorkflowStatus: 'CREATED',
//       createdAt: 2023-09-22T14:49:26.427Z,
//       imageSetId: 'xxxxxxxxxxxxxxxxxxxxxxxx',
//       imageSetState: 'ACTIVE',
//       versionId: '1'
//     }
//   ]
// }
return imageSetPropertiesList;
};

```

- Einzelheiten zur API finden Sie [ListImageSetVersions](#) in der AWS SDK für JavaScript API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

ListTagsForResource

Das folgende Codebeispiel zeigt die Verwendung `ListTagsForResource`.

SDK für JavaScript (v3)

```
import { ListTagsForResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 * or image set.
 */
export const listTagsForResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:abc:datastore/def/imageset/ghi",
) => {
  const response = await medicalImagingClient.send(
    new ListTagsForResourceCommand({ resourceArn: resourceArn }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '008fc6d3-abec-4870-a155-20fa3631e645',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   tags: { Deployment: 'Development' }
  // }

  return response;
};
```

- Einzelheiten zur API finden Sie [ListTagsForResource](#) in der AWS SDK für JavaScript API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

SearchImageSets

Das folgende Codebeispiel zeigt die Verwendung `SearchImageSets`.

SDK für JavaScript (v3)

Die Hilfsfunktion für die Suche nach Bildsätzen.

```
import { paginateSearchImageSets } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The data store's ID.
 * @param { import('@aws-sdk/client-medical-imaging').SearchFilter[] } filters - The
search criteria filters.
 * @param { import('@aws-sdk/client-medical-imaging').Sort } sort - The search
criteria sort.
 */
export const searchImageSets = async (
  datastoreId = "xxxxxxxx",
  searchCriteria = {},
) => {
  const paginatorConfig = {
    client: medicalImagingClient,
    pageSize: 50,
  };

  const commandParams = {
    datastoreId: datastoreId,
    searchCriteria: searchCriteria,
  };

  const paginator = paginateSearchImageSets(paginatorConfig, commandParams);

  const imageSetsMetadataSummaries = [];
  for await (const page of paginator) {
```

```

    // Each page contains a list of `jobSummaries`. The list is truncated if is
    // larger than `pageSize`.
    imageSetsMetadataSummaries.push(...page.imageSetsMetadataSummaries);
    console.log(page);
  }
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'f009ea9c-84ca-4749-b5b6-7164f00a5ada',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   imageSetsMetadataSummaries: [
  //     {
  //       DICOMTags: [Object],
  //       createdAt: "2023-09-19T16:59:40.551Z",
  //       imageSetId: '7f75e1b5c0f40eac2b24cf712f485f50',
  //       updatedAt: "2023-09-19T16:59:40.551Z",
  //       version: 1
  //     }
  //   ]
  // }

  return imageSetsMetadataSummaries;
};

```

Anwendungsfall 1: EQUAL-Operator.

```

const datastoreId = "12345678901234567890123456789012";

try {
  const searchCriteria = {
    filters: [
      {
        values: [{ DICOMPatientId: "1234567" }],
        operator: "EQUAL",
      },
    ],
  };

  await searchImageSets(datastoreId, searchCriteria);
}

```

```
} catch (err) {  
  console.error(err);  
}
```

Anwendungsfall #2: BETWEEN-Operator mit DICOMStudy Datum und DICOMStudy Uhrzeit.

```
const datastoreId = "12345678901234567890123456789012";  
  
try {  
  const searchCriteria = {  
    filters: [  
      {  
        values: [  
          {  
            DICOMStudyDateAndTime: {  
              DICOMStudyDate: "19900101",  
              DICOMStudyTime: "000000",  
            },  
          },  
          {  
            DICOMStudyDateAndTime: {  
              DICOMStudyDate: "20230901",  
              DICOMStudyTime: "000000",  
            },  
          },  
        ],  
        operator: "BETWEEN",  
      },  
    ],  
  };  
  
  await searchImageSets(datastoreId, searchCriteria);  
} catch (err) {  
  console.error(err);  
}
```

Anwendungsfall 3: BETWEEN-Operator mit createdAt. Zeitstudien wurden bisher fortgeführt.

```
const datastoreId = "12345678901234567890123456789012";  
  
try {
```

```
const searchCriteria = {
  filters: [
    {
      values: [
        { createdAt: new Date("1985-04-12T23:20:50.52Z") },
        { createdAt: new Date() },
      ],
      operator: "BETWEEN",
    },
  ],
};

await searchImageSets(datastoreId, searchCriteria);
} catch (err) {
  console.error(err);
}
```

Anwendungsfall #4: EQUAL-Operator für DICOMSeries instanceUID und BETWEEN für updatedAt und sortiere die Antwort in ASC-Reihenfolge für das updatedAt-Feld.

```
const datastoreId = "12345678901234567890123456789012";

try {
  const searchCriteria = {
    filters: [
      {
        values: [
          { updatedAt: new Date("1985-04-12T23:20:50.52Z") },
          { updatedAt: new Date() },
        ],
        operator: "BETWEEN",
      },
      {
        values: [
          {
            DICOMSeriesInstanceUID:
              "1.1.123.123456.1.12.1.1234567890.1234.12345678.123",
          },
        ],
        operator: "EQUAL",
      },
    ],
  },
```

```

        sort: {
            sortOrder: "ASC",
            sortField: "updatedAt",
        },
    };

    await searchImageSets(datastoreId, searchCriteria);
} catch (err) {
    console.error(err);
}

```

- Einzelheiten zur API finden Sie in der API-Referenz. [SearchImageSets](#) AWS SDK für JavaScript

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

StartDICOMImportJob

Das folgende Codebeispiel zeigt die Verwendung `StartDICOMImportJob`.

SDK für JavaScript (v3)

```

import { StartDICOMImportJobCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} jobName - The name of the import job.
 * @param {string} datastoreId - The ID of the data store.
 * @param {string} dataAccessRoleArn - The Amazon Resource Name (ARN) of the role
 * that grants permission.
 * @param {string} inputS3Uri - The URI of the S3 bucket containing the input files.
 * @param {string} outputS3Uri - The URI of the S3 bucket where the output files are
 * stored.
 */
export const startDicomImportJob = async (
    jobName = "test-1",
    datastoreId = "12345678901234567890123456789012",
    dataAccessRoleArn = "arn:aws:iam::xxxxxxxxxxxx:role/ImportJobDataAccessRole",
    inputS3Uri = "s3://medical-imaging-dicom-input/dicom_input/",

```

```

    outputS3Uri = "s3://medical-imaging-output/job_output/",
  ) => {
    const response = await medicalImagingClient.send(
      new StartDICOMImportJobCommand({
        jobName: jobName,
        datastoreId: datastoreId,
        dataAccessRoleArn: dataAccessRoleArn,
        inputS3Uri: inputS3Uri,
        outputS3Uri: outputS3Uri,
      }),
    );
    console.log(response);
    // {
    //   '$metadata': {
    //     httpStatusCode: 200,
    //     requestId: '6e81d191-d46b-4e48-a08a-cdcc7e11eb79',
    //     extendedRequestId: undefined,
    //     cfId: undefined,
    //     attempts: 1,
    //     totalRetryDelay: 0
    //   },
    //   datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
    //   jobId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
    //   jobStatus: 'SUBMITTED',
    //   submittedAt: 2023-09-22T14:48:45.767Z
    // }
    return response;
  };

```

- Einzelheiten zur API finden Sie unter [DICOMImportJob starten](#) in der AWS SDK für JavaScript API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

TagResource

Das folgende Codebeispiel zeigt die Verwendung TagResource.

SDK für JavaScript (v3)

```
import { TagResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 * or image set.
 * @param {Record<string,string>} tags - The tags to add to the resource as JSON.
 * - For example: {"Deployment" : "Development"}
 */
export const tagResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/imageset/
xxx",
  tags = {},
) => {
  const response = await medicalImagingClient.send(
    new TagResourceCommand({ resourceArn: resourceArn, tags: tags }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 204,
  //     requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }

  return response;
};
```

- Einzelheiten zur API finden Sie [TagResource](#) in der AWS SDK für JavaScript API-Referenz.

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

UntagResource

Das folgende Codebeispiel zeigt die Verwendung `UntagResource`.

SDK für JavaScript (v3)

```
import { UntagResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 * or image set.
 * @param {string[]} tagKeys - The keys of the tags to remove.
 */
export const untagResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/imageset/
xxx",
  tagKeys = [],
) => {
  const response = await medicalImagingClient.send(
    new UntagResourceCommand({ resourceArn: resourceArn, tagKeys: tagKeys }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 204,
  //     requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }

  return response;
};
```

- Einzelheiten zur API finden Sie [UntagResource](#) in der AWS SDK für JavaScript API-Referenz.

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

UpdateImageSetMetadata

Das folgende Codebeispiel zeigt die Verwendung `UpdateImageSetMetadata`.

SDK für JavaScript (v3)

```
import { UpdateImageSetMetadataCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} datastoreId - The ID of the HealthImaging data store.
 * @param {string} imageSetId - The ID of the HealthImaging image set.
 * @param {string} latestVersionId - The ID of the HealthImaging image set version.
 * @param {{}} updateMetadata - The metadata to update.
 * @param {boolean} force - Force the update.
 */
export const updateImageSetMetadata = async (
  datastoreId = "xxxxxxxxxx",
  imageSetId = "xxxxxxxxxx",
  latestVersionId = "1",
  updateMetadata = "{}",
  force = false,
) => {
  try {
    const response = await medicalImagingClient.send(
      new UpdateImageSetMetadataCommand({
        datastoreId: datastoreId,
        imageSetId: imageSetId,
        latestVersionId: latestVersionId,
        updateImageSetMetadataUpdates: updateMetadata,
        force: force,
      }),
    );
    console.log(response);
    // {
    //   '$metadata': {
    //     httpStatusCode: 200,
```

```

    //      requestId: '7966e869-e311-4bff-92ec-56a61d3003ea',
    //      extendedRequestId: undefined,
    //      cfId: undefined,
    //      attempts: 1,
    //      totalRetryDelay: 0
    // },
    //      createdAt: 2023-09-22T14:49:26.427Z,
    //      datastoreId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
    //      imageSetId: 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
    //      imageSetState: 'LOCKED',
    //      imageSetWorkflowStatus: 'UPDATING',
    //      latestVersionId: '4',
    //      updatedAt: 2023-09-27T19:41:43.494Z
    // }
    return response;
  } catch (err) {
    console.error(err);
  }
};

```

Anwendungsfall 1: Fügen Sie ein Attribut ein oder aktualisieren Sie es und erzwingen Sie das Update.

```

const insertAttributes = JSON.stringify({
  SchemaVersion: 1.1,
  Study: {
    DICOM: {
      StudyDescription: "CT CHEST",
    },
  },
});

const updateMetadata = {
  DICOMUpdates: {
    updatableAttributes: new TextEncoder().encode(insertAttributes),
  },
};

await updateImageSetMetadata(
  datastoreID,
  imageSetID,
  versionID,

```

```
    updateMetadata,  
    true,  
  );
```

Anwendungsfall 2: Entfernen Sie ein Attribut.

```
// Attribute key and value must match the existing attribute.  
const remove_attribute = JSON.stringify({  
  SchemaVersion: 1.1,  
  Study: {  
    DICOM: {  
      StudyDescription: "CT CHEST",  
    },  
  },  
});  
  
const updateMetadata = {  
  DICOMUpdates: {  
    removableAttributes: new TextEncoder().encode(remove_attribute),  
  },  
};  
  
await updateImageSetMetadata(  
  datastoreID,  
  imageSetID,  
  versionID,  
  updateMetadata,  
);
```

Anwendungsfall 3: Entfernen Sie eine Instance.

```
const remove_instance = JSON.stringify({  
  SchemaVersion: 1.1,  
  Study: {  
    Series: {  
      "1.1.1.1.1.1.12345.123456789012.123.12345678901234.1": {  
        Instances: {  
          "1.1.1.1.1.1.12345.123456789012.123.12345678901234.1": {},  
        },  
      },  
    },  
  },  
});
```

```
    },  
  });  
  
  const updateMetadata = {  
    DICOMUpdates: {  
      removableAttributes: new TextEncoder().encode(remove_instance),  
    },  
  };  
  
  await updateImageSetMetadata(  
    datastoreID,  
    imageSetID,  
    versionID,  
    updateMetadata,  
  );
```

Anwendungsfall 4: Kehren Sie zu einer früheren Version zurück.

```
const updateMetadata = {  
  revertToVersionId: "1",  
};  
  
await updateImageSetMetadata(  
  datastoreID,  
  imageSetID,  
  versionID,  
  updateMetadata,  
);
```

- Einzelheiten zur API finden Sie [UpdateImageSetMetadata](#) in der AWS SDK für JavaScript API-Referenz.

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Szenarien

Erste Schritte mit Bildsätzen und Bild-Frames

Das folgende Codebeispiel zeigt, wie DICOM-Dateien importiert und Bildrahmen heruntergeladen werden. HealthImaging

Die Implementierung ist als Befehlszeilenanwendung strukturiert.

- Richten Sie Ressourcen für einen DICOM-Import ein.
- Importieren Sie DICOM-Dateien in einen Datenspeicher.
- Rufen Sie den Bildsatz IDs für den Importauftrag ab.
- Rufen Sie den Bildrahmen IDs für die Bildsätze ab.
- Laden Sie die Bild-Frames herunter, dekodieren Sie sie und überprüfen Sie sie.
- Bereinigen Sie die Ressourcen.

SDK für JavaScript (v3)

Orchestrieren Sie die Schritte (index.js).

```
import {
  parseScenarioArgs,
  Scenario,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import {
  saveState,
  loadState,
} from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";

import {
  createStack,
  deployStack,
  getAccountId,
  getDatastoreName,
  getStackName,
  outputState,
  waitForStackCreation,
} from "../deploy-steps.js";
import {
  doCopy,
```

```
    selectDataset,
    copyDataset,
    outputCopiedObjects,
} from "./dataset-steps.js";
import {
    doImport,
    outputImportJobStatus,
    startDICOMImport,
    waitForImportJobCompletion,
} from "./import-steps.js";
import {
    getManifestFile,
    outputImageSetIds,
    parseManifestFile,
} from "./image-set-steps.js";
import {
    getImageSetMetadata,
    outputImageFrameIds,
} from "./image-frame-steps.js";
import { decodeAndVerifyImages, doVerify } from "./verify-steps.js";
import {
    confirmCleanup,
    deleteImageSets,
    deleteStack,
} from "./clean-up-steps.js";

const context = {};

const scenarios = {
    deploy: new Scenario(
        "Deploy Resources",
        [
            deployStack,
            getStackName,
            getDatastoreName,
            getAccountId,
            createStack,
            waitForStackCreation,
            outputState,
            saveState,
        ],
        context,
    ),
    demo: new Scenario(
```

```

    "Run Demo",
    [
        loadState,
        doCopy,
        selectDataset,
        copyDataset,
        outputCopiedObjects,
        doImport,
        startDICOMImport,
        waitForImportJobCompletion,
        outputImportJobStatus,
        getManifestFile,
        parseManifestFile,
        outputImageSetIds,
        getImageSetMetadata,
        outputImageFrameIds,
        doVerify,
        decodeAndVerifyImages,
        saveState,
    ],
    context,
),
destroy: new Scenario(
    "Clean Up Resources",
    [loadState, confirmCleanup, deleteImageSets, deleteStack],
    context,
),
};

// Call function if run directly
import { fileURLToPath } from "node:url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
    parseScenarioArgs(scenarios, {
        name: "Health Imaging Workflow",
        description:
            "Work with DICOM images using an AWS Health Imaging data store.",
        synopsis:
            "node index.js --scenario <deploy | demo | destroy> [-h|--help] [-y|--yes] [-v|--verbose]",
    });
}

```


Stellen Sie Ressourcen bereit (deploy-steps.js).

```
import fs from "node:fs/promises";
import path from "node:path";

import {
  CloudFormationClient,
  CreateStackCommand,
  DescribeStacksCommand,
} from "@aws-sdk/client-cloudformation";
import { STSClient, GetCallerIdentityCommand } from "@aws-sdk/client-sts";

import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

const cfnClient = new CloudFormationClient({});
const stsClient = new STSClient({});

const __dirname = path.dirname(new URL(import.meta.url).pathname);
const cfnTemplatePath = path.join(
  __dirname,
  "../../../../../scenarios/features/healthimaging_image_sets/resources/
cfn_template.yaml",
);

export const deployStack = new ScenarioInput(
  "deployStack",
  "Do you want to deploy the CloudFormation stack?",
  { type: "confirm" },
);

export const getStackName = new ScenarioInput(
  "getStackName",
  "Enter a name for the CloudFormation stack:",
  { type: "input", skipWhen: (/** @type {} */ state) => !state.deployStack },
);

export const getDatastoreName = new ScenarioInput(
  "getDatastoreName",
  "Enter a name for the HealthImaging datastore:",
```

```
    { type: "input", skipWhen: (/** @type {} */ state) => !state.deployStack },
  );

export const getAccountId = new ScenarioAction(
  "getAccountId",
  async (/** @type {} */ state) => {
    const command = new GetCallerIdentityCommand({});
    const response = await stsClient.send(command);
    state.accountId = response.Account;
  },
  {
    skipWhen: (/** @type {} */ state) => !state.deployStack,
  },
);

export const createStack = new ScenarioAction(
  "createStack",
  async (/** @type {} */ state) => {
    const stackName = state.getStackName;
    const datastoreId = state.getDatastoreId;
    const accountId = state.accountId;

    const command = new CreateStackCommand({
      StackName: stackName,
      TemplateBody: await fs.readFile(cfnTemplatePath, "utf8"),
      Capabilities: ["CAPABILITY_IAM"],
      Parameters: [
        {
          ParameterKey: "datastoreId",
          ParameterValue: datastoreId,
        },
        {
          ParameterKey: "userAccountId",
          ParameterValue: accountId,
        },
      ],
    });

    const response = await cfnClient.send(command);
    state.stackId = response.StackId;
  },
  { skipWhen: (/** @type {} */ state) => !state.deployStack },
);
```

```

export const waitForStackCreation = new ScenarioAction(
  "waitForStackCreation",
  async (/** @type {} */ state) => {
    const command = new DescribeStacksCommand({
      StackName: state.stackId,
    });

    await retry({ intervalInMs: 10000, maxRetries: 60 }, async () => {
      const response = await cfnClient.send(command);
      const stack = response.Stacks?.find(
        (s) => s.StackName === state.getStackName,
      );
      if (!stack || stack.StackStatus === "CREATE_IN_PROGRESS") {
        throw new Error("Stack creation is still in progress");
      }
      if (stack.StackStatus === "CREATE_COMPLETE") {
        state.stackOutputs = stack.Outputs?.reduce((acc, output) => {
          acc[output.OutputKey] = output.OutputValue;
          return acc;
        }, {});
      } else {
        throw new Error(
          `Stack creation failed with status: ${stack.StackStatus}`,
        );
      }
    });
  },
  {
    skipWhen: (/** @type {} */ state) => !state.deployStack,
  },
);

export const outputState = new ScenarioOutput(
  "outputState",
  (/** @type {} */ state) => {
    /**
     * @type {{ stackOutputs: { DatastoreID: string, BucketName: string, RoleArn:
     string }}}
     */
    const { stackOutputs } = state;
    return `Stack creation completed. Output values:
Datastore ID: ${stackOutputs?.DatastoreID}
Bucket Name: ${stackOutputs?.BucketName}
Role ARN: ${stackOutputs?.RoleArn}

```

```

    `;
  },
  { skipWhen: (/** @type {} */ state) => !state.deployStack },
);

```

Kopieren Sie DICOM-Dateien (dataset-steps.js).

```

import {
  S3Client,
  CopyObjectCommand,
  ListObjectsV2Command,
} from "@aws-sdk/client-s3";

import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

const s3Client = new S3Client({});

const datasetOptions = [
  {
    name: "CT of chest (2 images)",
    value: "00029d25-fb18-4d42-aaa5-a0897d1ac8f7",
  },
  {
    name: "CT of pelvis (57 images)",
    value: "00025d30-ef8f-4135-a35a-d83eff264fc1",
  },
  {
    name: "MRI of head (192 images)",
    value: "0002d261-8a5d-4e63-8e2e-0cbfac87b904",
  },
  {
    name: "MRI of breast (92 images)",
    value: "0002dd07-0b7f-4a68-a655-44461ca34096",
  },
];

/**
 * @typedef {{ stackOutputs: {

```

```
*   BucketName: string,
*   DatastoreID: string,
*   doCopy: boolean
* }}} State
*/

export const selectDataset = new ScenarioInput(
  "selectDataset",
  (state) => {
    if (!state.doCopy) {
      process.exit(0);
    }
    return "Select a DICOM dataset to import:";
  },
  {
    type: "select",
    choices: datasetOptions,
  },
);

export const doCopy = new ScenarioInput(
  "doCopy",
  "Do you want to copy images from the public dataset into your bucket?",
  {
    type: "confirm",
  },
);

export const copyDataset = new ScenarioAction(
  "copyDataset",
  async (/** @type { State } */ state) => {
    const inputBucket = state.stackOutputs.BucketName;
    const inputPrefix = "input/";
    const selectedDatasetId = state.selectDataset;

    const sourceBucket = "idc-open-data";
    const sourcePrefix = `${selectedDatasetId}`;

    const listObjectsCommand = new ListObjectsV2Command({
      Bucket: sourceBucket,
      Prefix: sourcePrefix,
    });

    const objects = await s3Client.send(listObjectsCommand);
```

```

const copyPromises = objects.Contents.map((object) => {
  const sourceKey = object.Key;
  const destinationKey = `${inputPrefix}${sourceKey}
    .split("/")
    .slice(1)
    .join("/")}`;

  const copyCommand = new CopyObjectCommand({
    Bucket: inputBucket,
    CopySource: `/${sourceBucket}/${sourceKey}`,
    Key: destinationKey,
  });

  return s3Client.send(copyCommand);
});

const results = await Promise.all(copyPromises);
state.copiedObjects = results.length;
},
);

export const outputCopiedObjects = new ScenarioOutput(
  "outputCopiedObjects",
  (state) => `${state.copiedObjects} DICOM files were copied.`,
);

```

Starten Sie den Import in den Datenspeicher (import-steps.js).

```

import {
  MedicalImagingClient,
  StartDICOMImportJobCommand,
  GetDICOMImportJobCommand,
} from "@aws-sdk/client-medical-imaging";

import {
  ScenarioAction,
  ScenarioOutput,
  ScenarioInput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

```

```

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }}} State
 */

export const doImport = new ScenarioInput(
  "doImport",
  "Do you want to import DICOM images into your datastore?",
  {
    type: "confirm",
    default: true,
  },
);

export const startDICOMImport = new ScenarioAction(
  "startDICOMImport",
  async (** @type {State} */ state) => {
    if (!state.doImport) {
      process.exit(0);
    }
    const medicalImagingClient = new MedicalImagingClient({});
    const inputS3Uri = `s3://${state.stackOutputs.BucketName}/input/`;
    const outputS3Uri = `s3://${state.stackOutputs.BucketName}/output/`;

    const command = new StartDICOMImportJobCommand({
      dataAccessRoleArn: state.stackOutputs.RoleArn,
      datastoreId: state.stackOutputs.DatastoreID,
      inputS3Uri,
      outputS3Uri,
    });

    const response = await medicalImagingClient.send(command);
    state.importJobId = response.jobId;
  },
);

export const waitForImportJobCompletion = new ScenarioAction(
  "waitForImportJobCompletion",
  async (** @type {State} */ state) => {
    const medicalImagingClient = new MedicalImagingClient({});
    const command = new GetDICOMImportJobCommand({

```

```

        datastoreId: state.stackOutputs.DatastoreID,
        jobId: state.importJobId,
    });

    await retry({ intervalInMs: 10000, maxRetries: 60 }, async () => {
        const response = await medicalImagingClient.send(command);
        const jobStatus = response.jobProperties?.jobStatus;
        if (!jobStatus || jobStatus === "IN_PROGRESS") {
            throw new Error("Import job is still in progress");
        }
        if (jobStatus === "COMPLETED") {
            state.importJobOutputS3Uri = response.jobProperties.outputS3Uri;
        } else {
            throw new Error(`Import job failed with status: ${jobStatus}`);
        }
    });
},
);

export const outputImportJobStatus = new ScenarioOutput(
    "outputImportJobStatus",
    (state) =>
        `DICOM import job completed. Output location: ${state.importJobOutputS3Uri}`,
);

```

Bilddatensatz abrufen IDs (image-set-steps.js-).

```

import { S3Client, GetObjectCommand } from "@aws-sdk/client-s3";

import {
    ScenarioAction,
    ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }, importJobId: string,
 * importJobOutputS3Uri: string,
 * imageSetIds: string[],

```



```

    * manifestContent: { jobSummary: { imageSetsSummary: { imageSetId: string }[] } }
    * }} State
    */

const s3Client = new S3Client({});

export const getManifestFile = new ScenarioAction(
  "getManifestFile",
  async (** @type {State} */ state) => {
    const bucket = state.stackOutputs.BucketName;
    const prefix = `output/${state.stackOutputs.DatastoreID}-DicomImport-
${state.importJobId}/`;
    const key = `${prefix}job-output-manifest.json`;

    const command = new GetObjectCommand({
      Bucket: bucket,
      Key: key,
    });

    const response = await s3Client.send(command);
    const manifestContent = await response.Body.transformToString();
    state.manifestContent = JSON.parse(manifestContent);
  },
);

export const parseManifestFile = new ScenarioAction(
  "parseManifestFile",
  (** @type {State} */ state) => {
    const imageSetIds =
      state.manifestContent.jobSummary.imageSetsSummary.reduce((ids, next) => {
        return Object.assign({}, ids, {
          [next.imageSetId]: next.imageSetId,
        });
      }, {});
    state.imageSetIds = Object.keys(imageSetIds);
  },
);

export const outputImageSetIds = new ScenarioOutput(
  "outputImageSetIds",
  (** @type {State} */ state) =>
    `The image sets created by this import job are: \n${state.imageSetIds
      .map((id) => `Image set: ${id}`)
      .join("\n")}``,
);

```

```
);
```

Holen Sie sich den Bildrahmen IDs (image-frame-steps.js).

```
import {
  MedicalImagingClient,
  GetImageSetMetadataCommand,
} from "@aws-sdk/client-medical-imaging";
import { gunzip } from "node:zlib";
import { promisify } from "node:util";

import {
  ScenarioAction,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

const gunzipAsync = promisify(gunzip);

/**
 * @typedef {Object} DICOMValueRepresentation
 * @property {string} name
 * @property {string} type
 * @property {string} value
 */

/**
 * @typedef {Object} ImageFrameInformation
 * @property {string} ID
 * @property {Array<{ Checksum: number, Height: number, Width: number }>}
  PixelDataChecksumFromBaseToFullResolution
 * @property {number} MinPixelValue
 * @property {number} MaxPixelValue
 * @property {number} FrameSizeInBytes
 */

/**
 * @typedef {Object} DICOMMetadata
 * @property {Object} DICOM
 * @property {DICOMValueRepresentation[]} DICOMVRs
 * @property {ImageFrameInformation[]} ImageFrames
 */
```

```
/**
 * @typedef {Object} Series
 * @property {{ [key: string]: DICOMMetadata }} Instances
 */

/**
 * @typedef {Object} Study
 * @property {Object} DICOM
 * @property {Series[]} Series
 */

/**
 * @typedef {Object} Patient
 * @property {Object} DICOM
 */

/**
 * @typedef {{
 *   SchemaVersion: string,
 *   DatastoreID: string,
 *   ImageSetID: string,
 *   Patient: Patient,
 *   Study: Study
 * }} ImageSetMetadata
 */

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }, imageSetIds: string[] }} State
 */

const medicalImagingClient = new MedicalImagingClient({});

export const getImageSetMetadata = new ScenarioAction(
  "getImageSetMetadata",
  async (** @type {State} */ state) => {
    const outputMetadata = [];

    for (const imageSetId of state.imageSetIds) {
      const command = new GetImageSetMetadataCommand({
        datastoreId: state.stackOutputs.DatastoreID,
```

```

        imageSetId,
    });

    const response = await medicalImagingClient.send(command);
    const compressedMetadataBlob =
        await response.imageSetMetadataBlob.transformToByteArray();
    const decompressedMetadata = await gunzipAsync(compressedMetadataBlob);
    const imageSetMetadata = JSON.parse(decompressedMetadata.toString());

    outputMetadata.push(imageSetMetadata);
}

state.imageSetMetadata = outputMetadata;
},
);

export const outputImageFrameIds = new ScenarioOutput(
    "outputImageFrameIds",
    (/** @type {State & { imageSetMetadata: ImageSetMetadata[] }} */ state) => {
        let output = "";

        for (const metadata of state.imageSetMetadata) {
            const imageSetId = metadata.ImageSetID;
            /** @type {DICOMMetadata[]} */
            const instances = Object.values(metadata.Study.Series).flatMap(
                (series) => {
                    return Object.values(series.Instances);
                },
            );
            const imageFrameIds = instances.flatMap((instance) =>
                instance.ImageFrames.map((frame) => frame.ID),
            );

            output += `Image set ID: ${imageSetId}\nImage frame IDs:\n
${imageFrameIds.join(
                "\n",
            )}\n\n`;
        }

        return output;
    },
);

```

Überprüfen Sie die Image-Rahmen (verify-steps.js). Die Bibliothek [zur Überprüfung von AWS HealthImaging Pixeldaten](#) wurde zur Überprüfung verwendet.

```
import { spawn } from "node:child_process";

import {
  ScenarioAction,
  ScenarioInput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * @typedef {Object} DICOMValueRepresentation
 * @property {string} name
 * @property {string} type
 * @property {string} value
 */

/**
 * @typedef {Object} ImageFrameInformation
 * @property {string} ID
 * @property {Array<{ Checksum: number, Height: number, Width: number }>}
  PixelDataChecksumFromBaseToFullResolution
 * @property {number} MinPixelValue
 * @property {number} MaxPixelValue
 * @property {number} FrameSizeInBytes
 */

/**
 * @typedef {Object} DICOMMetadata
 * @property {Object} DICOM
 * @property {DICOMValueRepresentation[]} DICOMVRs
 * @property {ImageFrameInformation[]} ImageFrames
 */

/**
 * @typedef {Object} Series
 * @property {{ [key: string]: DICOMMetadata }} Instances
 */

/**
 * @typedef {Object} Study
 * @property {Object} DICOM
 * @property {Series[]} Series
```

```

 */

/**
 * @typedef {Object} Patient
 * @property {Object} DICOM
 */

/**
 * @typedef {{
 *   SchemaVersion: string,
 *   DatastoreID: string,
 *   ImageSetID: string,
 *   Patient: Patient,
 *   Study: Study
 * }} ImageSetMetadata
 */

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }, imageSetMetadata: ImageSetMetadata[] }} State
 */

export const doVerify = new ScenarioInput(
  "doVerify",
  "Do you want to verify the imported images?",
  {
    type: "confirm",
    default: true,
  },
);

export const decodeAndVerifyImages = new ScenarioAction(
  "decodeAndVerifyImages",
  async (** @type {State} */ state) => {
    if (!state.doVerify) {
      process.exit(0);
    }
    const verificationTool = "./pixel-data-verification/index.js";

    for (const metadata of state.imageSetMetadata) {
      const datastoreId = state.stackOutputs.DatastoreID;

```

```

const imageSetId = metadata.ImageSetID;

for (const [seriesInstanceId, series] of Object.entries(
  metadata.Study.Series,
)) {
  for (const [sopInstanceId, _] of Object.entries(series.Instances)) {
    console.log(
      `Verifying image set ${imageSetId} with series ${seriesInstanceId} and
sop ${sopInstanceId}`,
    );
    const child = spawn(
      "node",
      [
        verificationTool,
        datastoreId,
        imageSetId,
        seriesInstanceId,
        sopInstanceId,
      ],
      { stdio: "inherit" },
    );

    await new Promise((resolve, reject) => {
      child.on("exit", (code) => {
        if (code === 0) {
          resolve();
        } else {
          reject(
            new Error(
              `Verification tool exited with code ${code} for image set
${imageSetId}`,
            ),
          );
        }
      });
    });
  }
}
},
);

```

Vernichten Sie Ressourcen (clean-up-steps.js).

```
import {
  CloudFormationClient,
  DeleteStackCommand,
} from "@aws-sdk/client-cloudformation";
import {
  MedicalImagingClient,
  DeleteImageSetCommand,
} from "@aws-sdk/client-medical-imaging";

import {
  ScenarioAction,
  ScenarioInput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * @typedef {Object} DICOMValueRepresentation
 * @property {string} name
 * @property {string} type
 * @property {string} value
 */

/**
 * @typedef {Object} ImageFrameInformation
 * @property {string} ID
 * @property {Array<{ Checksum: number, Height: number, Width: number }>}
  PixelDataChecksumFromBaseToFullResolution
 * @property {number} MinPixelValue
 * @property {number} MaxPixelValue
 * @property {number} FrameSizeInBytes
 */

/**
 * @typedef {Object} DICOMMetadata
 * @property {Object} DICOM
 * @property {DICOMValueRepresentation[]} DICOMVRs
 * @property {ImageFrameInformation[]} ImageFrames
 */

/**
 * @typedef {Object} Series
 * @property {{ [key: string]: DICOMMetadata }} Instances
 */
```



```
/**
 * @typedef {Object} Study
 * @property {Object} DICOM
 * @property {Series[]} Series
 */

/**
 * @typedef {Object} Patient
 * @property {Object} DICOM
 */

/**
 * @typedef {{
 *   SchemaVersion: string,
 *   DatastoreID: string,
 *   ImageSetID: string,
 *   Patient: Patient,
 *   Study: Study
 * }} ImageSetMetadata
 */

/**
 * @typedef {{ stackOutputs: {
 *   BucketName: string,
 *   DatastoreID: string,
 *   RoleArn: string
 * }, imageSetMetadata: ImageSetMetadata[] }} State
 */

const cfnClient = new CloudFormationClient({});
const medicalImagingClient = new MedicalImagingClient({});

export const confirmCleanup = new ScenarioInput(
  "confirmCleanup",
  "Do you want to delete the created resources?",
  { type: "confirm" },
);

export const deleteImageSets = new ScenarioAction(
  "deleteImageSets",
  async (** @type {State} */ state) => {
    const datastoreId = state.stackOutputs.DatastoreID;
```

```

    for (const metadata of state.imageSetMetadata) {
      const command = new DeleteImageSetCommand({
        datastoreId,
        imageSetId: metadata.ImageSetID,
      });

      try {
        await medicalImagingClient.send(command);
        console.log(`Successfully deleted image set ${metadata.ImageSetID}`);
      } catch (e) {
        if (e instanceof Error) {
          if (e.name === "ConflictException") {
            console.log(`Image set ${metadata.ImageSetID} already deleted`);
          }
        }
      }
    },
    {
      skipWhen: (/** @type {} */ state) => !state.confirmCleanup,
    },
  );

export const deleteStack = new ScenarioAction(
  "deleteStack",
  async (/** @type {State} */ state) => {
    const stackName = state.getStackName;

    const command = new DeleteStackCommand({
      StackName: stackName,
    });

    await cfnClient.send(command);
    console.log(`Stack ${stackName} deletion initiated`);
  },
  {
    skipWhen: (/** @type {} */ state) => !state.confirmCleanup,
  },
);

```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für JavaScript - API-Referenz.

- [DeleteImageSet](#)
- [DICOMImportJob bekommen](#)
- [GetImageFrame](#)
- [GetImageSetMetadata](#)
- [SearchImageSets](#)
- [DICOMImportJob starten](#)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Taggen eines Datenspeichers

Das folgende Codebeispiel zeigt, wie ein HealthImaging Datenspeicher markiert wird.

SDK für JavaScript (v3)

So taggen Sie einen Datenspeicher.

```
try {
  const datastoreArn =
    "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012";
  const tags = {
    Deployment: "Development",
  };
  await tagResource(datastoreArn, tags);
} catch (e) {
  console.log(e);
}
```

Die Hilfsfunktion zum Taggen einer Ressource.

```
import { TagResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
```

```

* @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
or image set.
* @param {Record<string,string>} tags - The tags to add to the resource as JSON.
*
*   - For example: {"Deployment" : "Development"}
*/
export const tagResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/imageset/
xxx",
  tags = {},
) => {
  const response = await medicalImagingClient.send(
    new TagResourceCommand({ resourceArn: resourceArn, tags: tags }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 204,
  //     requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }

  return response;
};

```

So listen Sie Tags für einen Datenspeicher auf.

```

try {
  const datastoreArn =
    "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012";
  const { tags } = await listTagsForResource(datastoreArn);
  console.log(tags);
} catch (e) {
  console.log(e);
}

```

Die Hilfsfunktion zum Auflisten von Tags einer Ressource.

```

import { ListTagsForResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 * or image set.
 */
export const listTagsForResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:abc:datastore/def/imageset/ghi",
) => {
  const response = await medicalImagingClient.send(
    new ListTagsForResourceCommand({ resourceArn: resourceArn }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '008fc6d3-abec-4870-a155-20fa3631e645',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   tags: { Deployment: 'Development' }
  // }

  return response;
};

```

So entfernen Sie Tags von einem Datenspeicher.

```

try {
  const datastoreArn =
    "arn:aws:medical-imaging:us-east-1:123456789012:datastore/12345678901234567890123456789012";
  const keys = ["Deployment"];
  await untagResource(datastoreArn, keys);
} catch (e) {
  console.log(e);
}

```

Die Hilfsfunktion zum Entfernen von Tags von einer Ressource.

```
import { UntagResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 * or image set.
 * @param {string[]} tagKeys - The keys of the tags to remove.
 */
export const untagResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/imageset/
xxx",
  tagKeys = [],
) => {
  const response = await medicalImagingClient.send(
    new UntagResourceCommand({ resourceArn: resourceArn, tagKeys: tagKeys }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 204,
  //     requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }

  return response;
};
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für JavaScript - API-Referenz.
 - [ListTagsForResource](#)
 - [TagResource](#)
 - [UntagResource](#)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Taggen eines Imagesatzes

Das folgende Codebeispiel zeigt, wie ein HealthImaging Bilddatensatz markiert wird.

SDK für JavaScript (v3)

So taggen Sie einen Imagesatz.

```
try {
  const imagesetArn =
    "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012/
imageset/12345678901234567890123456789012";
  const tags = {
    Deployment: "Development",
  };
  await tagResource(imagesetArn, tags);
} catch (e) {
  console.log(e);
}
```

Die Hilfsfunktion zum Taggen einer Ressource.

```
import { TagResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 * or image set.
 * @param {Record<string,string>} tags - The tags to add to the resource as JSON.
 * - For example: {"Deployment" : "Development"}
 */
export const tagResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/imageset/
xxx",
```

```

    tags = {},
  ) => {
    const response = await medicalImagingClient.send(
      new TagResourceCommand({ resourceArn: resourceArn, tags: tags }),
    );
    console.log(response);
    // {
    //   '$metadata': {
    //     httpStatusCode: 204,
    //     requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
    //     extendedRequestId: undefined,
    //     cfId: undefined,
    //     attempts: 1,
    //     totalRetryDelay: 0
    //   }
    // }

    return response;
  };

```

So listen Sie Tags für einen Imagesatz auf.

```

try {
  const imagesetArn =
    "arn:aws:medical-imaging:us-
east-1:123456789012:datastore/12345678901234567890123456789012/
imageset/12345678901234567890123456789012";
  const { tags } = await listTagsForResource(imagesetArn);
  console.log(tags);
} catch (e) {
  console.log(e);
}

```

Die Hilfsfunktion zum Auflisten von Tags einer Ressource.

```

import { ListTagsForResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 * or image set.

```



```

*/
export const listTagsForResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:abc:datastore/def/imageset/ghi",
) => {
  const response = await medicalImagingClient.send(
    new ListTagsForResourceCommand({ resourceArn: resourceArn }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '008fc6d3-abec-4870-a155-20fa3631e645',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   tags: { Deployment: 'Development' }
  // }

  return response;
};

```

So entfernen Sie Tags von einem Imagesatz.

```

try {
  const imagesetArn =
    "arn:aws:medical-imaging:us-east-1:123456789012:datastore/12345678901234567890123456789012/imageset/12345678901234567890123456789012";
  const keys = ["Deployment"];
  await untagResource(imagesetArn, keys);
} catch (e) {
  console.log(e);
}

```

Die Hilfsfunktion zum Entfernen von Tags von einer Ressource.

```

import { UntagResourceCommand } from "@aws-sdk/client-medical-imaging";
import { medicalImagingClient } from "../libs/medicalImagingClient.js";

```

```
/**
 * @param {string} resourceArn - The Amazon Resource Name (ARN) for the data store
 * or image set.
 * @param {string[]} tagKeys - The keys of the tags to remove.
 */
export const untagResource = async (
  resourceArn = "arn:aws:medical-imaging:us-east-1:xxxxxx:datastore/xxxxx/imageset/
xxx",
  tagKeys = [],
) => {
  const response = await medicalImagingClient.send(
    new UntagResourceCommand({ resourceArn: resourceArn, tagKeys: tagKeys }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 204,
  //     requestId: '8a6de9a3-ec8e-47ef-8643-473518b19d45',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }

  return response;
};
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für JavaScript - API-Referenz.
 - [ListTagsForResource](#)
 - [TagResource](#)
 - [UntagResource](#)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

IAM-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK für JavaScript (v3) mit IAM Aktionen ausführen und allgemeine Szenarien implementieren.

Bei Grundlagen handelt es sich um Codebeispiele, die Ihnen zeigen, wie Sie die wesentlichen Vorgänge innerhalb eines Services ausführen.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien anzeigen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb eines Service aufrufen oder mit anderen AWS-Services kombinieren.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kodex finden.

Themen

- [Erste Schritte](#)
- [Grundlagen](#)
- [Aktionen](#)
- [Szenarien](#)

Erste Schritte

Hello IAM

Das folgende Codebeispiel zeigt, wie Sie mit IAM beginnen.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { IAMClient, paginateListPolicies } from "@aws-sdk/client-iam";
```

```
const client = new IAMClient({});

export const listLocalPolicies = async () => {
  /**
   * In v3, the clients expose paginateOperationName APIs that are written using
   * async generators so that you can use async iterators in a for await..of loop.
   * https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/index.html#paginators
   */
  const paginator = paginateListPolicies(
    { client, pageSize: 10 },
    // List only customer managed policies.
    { Scope: "Local" },
  );

  console.log("IAM policies defined in your account:");
  let policyCount = 0;
  for await (const page of paginator) {
    if (page.Policies) {
      for (const policy of page.Policies) {
        console.log(`${policy.PolicyName}`);
        policyCount++;
      }
    }
  }
  console.log(`Found ${policyCount} policies.`);
};
```

- Einzelheiten zur API finden Sie [ListPolicies](#) in der AWS SDK für JavaScript API-Referenz.

Grundlagen

Kennenlernen der Grundlagen

Das folgende Codebeispiel veranschaulicht, wie Sie einen Benutzer erstellen und eine Rolle annehmen lassen.

Warning

Um Sicherheitsrisiken zu vermeiden, sollten Sie IAM-Benutzer nicht zur Authentifizierung verwenden, wenn Sie speziell entwickelte Software entwickeln oder mit echten Daten

arbeiten. Verwenden Sie stattdessen den Verbund mit einem Identitätsanbieter wie [AWS IAM Identity Center](#).

- Erstellen Sie einen Benutzer ohne Berechtigungen.
- Erstellen einer Rolle, die die Berechtigung zum Auflisten von Amazon-S3-Buckets für das Konto erteilt.
- Hinzufügen einer Richtlinie, damit der Benutzer die Rolle übernehmen kann.
- Übernehmen Sie die Rolle und listen Sie S3-Buckets mit temporären Anmeldeinformationen auf, und bereinigen Sie dann die Ressourcen.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie einen IAM-Benutzer und eine Rolle, die die Berechtigung zum Auflisten von Amazon-S3-Buckets erteilt. Der Benutzer hat nur Rechte, um die Rolle anzunehmen. Nachdem Sie die Rolle übernommen haben, verwenden Sie temporäre Anmeldeinformationen, um Buckets für das Konto aufzulisten.

```
import {
  CreateUserCommand,
  GetUserCommand,
  CreateAccessKeyCommand,
  CreatePolicyCommand,
  CreateRoleCommand,
  AttachRolePolicyCommand,
  DeleteAccessKeyCommand,
  DeleteUserCommand,
  DeleteRoleCommand,
  DeletePolicyCommand,
  DetachRolePolicyCommand,
  IAMClient,
} from "@aws-sdk/client-iam";
import { ListBucketsCommand, S3Client } from "@aws-sdk/client-s3";
```

```

import { AssumeRoleCommand, STSClient } from "@aws-sdk/client-sts";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";
import { ScenarioInput } from "@aws-doc-sdk-examples/lib/scenario/index.js";

// Set the parameters.
const iamClient = new IAMClient({});
const userName = "iam_basic_test_username";
const policyName = "iam_basic_test_policy";
const roleName = "iam_basic_test_role";

/**
 * Create a new IAM user. If the user already exists, give
 * the option to delete and re-create it.
 * @param {string} name
 */
export const createUser = async (name, confirmAll = false) => {
  try {
    const { User } = await iamClient.send(
      new GetUserCommand({ UserName: name }),
    );
    const input = new ScenarioInput(
      "deleteUser",
      "Do you want to delete and remake this user?",
      { type: "confirm" },
    );
    const deleteUser = await input.handle({}, { confirmAll });
    // If the user exists, and you want to delete it, delete the user
    // and then create it again.
    if (deleteUser) {
      await iamClient.send(new DeleteUserCommand({ UserName: User.UserName }));
      await iamClient.send(new CreateUserCommand({ UserName: name }));
    } else {
      console.warn(
        `${name} already exists. The scenario may not work as expected.`,
      );
      return User;
    }
  } catch (caught) {
    // If there is no user by that name, create one.
    if (caught instanceof Error && caught.name === "NoSuchEntityException") {
      const { User } = await iamClient.send(
        new CreateUserCommand({ UserName: name }),
      );
      return User;
    }
  }
}

```

```
    }
    throw caught;
  }
};

export const main = async (confirmAll = false) => {
  // Create a user. The user has no permissions by default.
  const User = await createUser(userName, confirmAll);

  if (!User) {
    throw new Error("User not created");
  }

  // Create an access key. This key is used to authenticate the new user to
  // Amazon Simple Storage Service (Amazon S3) and AWS Security Token Service (AWS
  STS).
  // It's not best practice to use access keys. For more information, see https://aws.amazon.com/iam/resources/best-practices/.
  const createAccessKeyResponse = await iamClient.send(
    new CreateAccessKeyCommand({ UserName: userName }),
  );

  if (
    !createAccessKeyResponse.AccessKey?.AccessKeyId ||
    !createAccessKeyResponse.AccessKey?.SecretAccessKey
  ) {
    throw new Error("Access key not created");
  }

  const {
    AccessKey: { AccessKeyId, SecretAccessKey },
  } = createAccessKeyResponse;

  let s3Client = new S3Client({
    credentials: {
      accessKeyId: AccessKeyId,
      secretAccessKey: SecretAccessKey,
    },
  });

  // Retry the list buckets operation until it succeeds. InvalidAccessKeyId is
  // thrown while the user and access keys are still stabilizing.
  await retry({ intervalInMs: 1000, maxRetries: 300 }, async () => {
    try {
```

```
    return await listBuckets(s3Client);
  } catch (err) {
    if (err instanceof Error && err.name === "InvalidAccessKeyId") {
      throw err;
    }
  }
});

// Retry the create role operation until it succeeds. A MalformedPolicyDocument
error
// is thrown while the user and access keys are still stabilizing.
const { Role } = await retry(
  {
    intervalInMs: 2000,
    maxRetries: 60,
  },
  () =>
    iamClient.send(
      new CreateRoleCommand({
        AssumeRolePolicyDocument: JSON.stringify({
          Version: "2012-10-17",
          Statement: [
            {
              Effect: "Allow",
              Principal: {
                // Allow the previously created user to assume this role.
                AWS: User.Arn,
              },
              Action: "sts:AssumeRole",
            },
          ],
        }),
        RoleName: roleName,
      }),
    ),
);

if (!Role) {
  throw new Error("Role not created");
}

// Create a policy that allows the user to list S3 buckets.
const { Policy: listBucketPolicy } = await iamClient.send(
  new CreatePolicyCommand({
```



```
PolicyDocument: JSON.stringify({
  Version: "2012-10-17",
  Statement: [
    {
      Effect: "Allow",
      Action: ["s3:ListAllMyBuckets"],
      Resource: "*",
    },
  ],
}),
PolicyName: policyName,
}),
);

if (!listBucketPolicy) {
  throw new Error("Policy not created");
}

// Attach the policy granting the 's3:ListAllMyBuckets' action to the role.
await iamClient.send(
  new AttachRolePolicyCommand({
    PolicyArn: listBucketPolicy.Arn,
    RoleName: Role.RoleName,
  }),
);

// Assume the role.
const stsClient = new STSClient({
  credentials: {
    accessKeyId: AccessKeyId,
    secretAccessKey: SecretAccessKey,
  },
});

// Retry the assume role operation until it succeeds.
const { Credentials } = await retry(
  { intervalInMs: 2000, maxRetries: 60 },
  () =>
    stsClient.send(
      new AssumeRoleCommand({
        RoleArn: Role.Arn,
        RoleSessionName: `iamBasicScenarioSession-${Math.floor(
          Math.random() * 1000000,
        )}`,
      })
    )
  )
```

```
        DurationSeconds: 900,
      })),
    ),
  );

if (!Credentials?.AccessKeyId || !Credentials?.SecretAccessKey) {
  throw new Error("Credentials not created");
}

s3Client = new S3Client({
  credentials: {
    accessKeyId: Credentials.AccessKeyId,
    secretAccessKey: Credentials.SecretAccessKey,
    sessionToken: Credentials.SessionToken,
  },
});

// List the S3 buckets again.
// Retry the list buckets operation until it succeeds. AccessDenied might
// be thrown while the role policy is still stabilizing.
await retry({ intervalInMs: 2000, maxRetries: 120 }, () =>
  listBuckets(s3Client),
);

// Clean up.
await iamClient.send(
  new DetachRolePolicyCommand({
    PolicyArn: listBucketPolicy.Arn,
    RoleName: Role.RoleName,
  }),
);

await iamClient.send(
  new DeletePolicyCommand({
    PolicyArn: listBucketPolicy.Arn,
  }),
);

await iamClient.send(
  new DeleteRoleCommand({
    RoleName: Role.RoleName,
  }),
);
```

```
await iamClient.send(
  new DeleteAccessKeyCommand({
    UserName: userName,
    AccessKeyId,
  }),
);

await iamClient.send(
  new DeleteUserCommand({
    UserName: userName,
  }),
);
};

/**
 *
 * @param {S3Client} s3Client
 */
const listBuckets = async (s3Client) => {
  const { Buckets } = await s3Client.send(new ListBucketsCommand({}));

  if (!Buckets) {
    throw new Error("Buckets not listed");
  }

  console.log(Buckets.map((bucket) => bucket.Name).join("\n"));
};
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für JavaScript - API-Referenz.
 - [AttachRolePolicy](#)
 - [CreateAccessKey](#)
 - [CreatePolicy](#)
 - [CreateRole](#)
 - [CreateUser](#)
 - [DeleteAccessKey](#)
 - [DeletePolicy](#)
 - [DeleteRole](#)

- [DeleteUser](#)
- [DeleteUserPolicy](#)
- [DetachRolePolicy](#)
- [PutUserPolicy](#)

Aktionen

AttachRolePolicy

Das folgende Codebeispiel zeigt die Verwendung `AttachRolePolicy`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Anfügen einer Richtlinie.

```
import { AttachRolePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} policyArn
 * @param {string} roleName
 */
export const attachRolePolicy = (policyArn, roleName) => {
  const command = new AttachRolePolicyCommand({
    PolicyArn: policyArn,
    RoleName: roleName,
  });

  return client.send(command);
};
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [AttachRolePolicy](#) in der AWS SDK für JavaScript API-Referenz.

CreateAccessKey

Das folgende Codebeispiel zeigt die Verwendung `CreateAccessKey`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie den Zugriffsschlüssel.

```
import { CreateAccessKeyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} userName
 */
export const createAccessKey = (userName) => {
  const command = new CreateAccessKeyCommand({ UserName: userName });
  return client.send(command);
};
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [CreateAccessKey](#) in der AWS SDK für JavaScript API-Referenz.

CreateAccountAlias

Das folgende Codebeispiel zeigt die Verwendung `CreateAccountAlias`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie den Konto-Alias.

```
import { CreateAccountAliasCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} alias - A unique name for the account alias.
 * @returns
 */
export const createAccountAlias = (alias) => {
  const command = new CreateAccountAliasCommand({
    AccountAlias: alias,
  });

  return client.send(command);
};
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [CreateAccountAlias](#) in der AWS SDK für JavaScript API-Referenz.

CreateGroup

Das folgende Codebeispiel zeigt die Verwendung `CreateGroup`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { CreateGroupCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} groupName
 */
export const createGroup = async (groupName) => {
  const command = new CreateGroupCommand({ GroupName: groupName });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Einzelheiten zur API finden Sie [CreateGroup](#) in der AWS SDK für JavaScript API-Referenz.

CreateInstanceProfile

Das folgende Codebeispiel zeigt die Verwendung `CreateInstanceProfile`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
const { InstanceProfile } = await iamClient.send(
```

```

    new CreateInstanceProfileCommand({
      InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
    }),
  );
  await waitUntilInstanceProfileExists(
    { client: iamClient },
    { InstanceProfileName: NAMES.ssmOnlyInstanceProfileName },
  );

```

- Einzelheiten zur API finden Sie [CreateInstanceProfile](#) in der AWS SDK für JavaScript API-Referenz.

CreatePolicy

Das folgende Codebeispiel zeigt die Verwendung `CreatePolicy`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie die Richtlinie.

```

import { CreatePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} policyName
 */
export const createPolicy = (policyName) => {
  const command = new CreatePolicyCommand({
    PolicyDocument: JSON.stringify({
      Version: "2012-10-17",
      Statement: [
        {

```



```

        Effect: "Allow",
        Action: "*",
        Resource: "*",
    },
],
}),
PolicyName: policyName,
});

return client.send(command);
};

```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [CreatePolicy](#) in der AWS SDK für JavaScript API-Referenz.

CreateRole

Das folgende Codebeispiel zeigt die Verwendung `CreateRole`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie die Rolle.

```

import { CreateRoleCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} roleName
 */
export const createRole = (roleName) => {
    const command = new CreateRoleCommand({
        AssumeRolePolicyDocument: JSON.stringify({

```

```

    Version: "2012-10-17",
    Statement: [
      {
        Effect: "Allow",
        Principal: {
          Service: "lambda.amazonaws.com",
        },
        Action: "sts:AssumeRole",
      },
    ],
  )),
  RoleName: roleName,
});

return client.send(command);
};

```

- Einzelheiten zur API finden Sie [CreateRole](#) in der AWS SDK für JavaScript API-Referenz.

CreateSAMLProvider

Das folgende Codebeispiel zeigt die Verwendung `CreateSAMLProvider`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```

import { CreateSAMLProviderCommand, IAMClient } from "@aws-sdk/client-iam";
import { readFileSync } from "node:fs";
import * as path from "node:path";
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";

const client = new IAMClient({});

/**
 * This sample document was generated using Auth0.

```

```

* For more information on generating this document,
  see https://docs.aws.amazon.com/IAM/latest/UserGuide/
  id_roles_providers_create_saml.html#samlstep1.
*/
const sampleMetadataDocument = readFileSync(
  path.join(
    dirnameFromMetaUrl(import.meta.url),
    "../../../../../resources/sample_files/sample_saml_metadata.xml",
  ),
);

/**
 *
 * @param {*} providerName
 * @returns
 */
export const createSAMLProvider = async (providerName) => {
  const command = new CreateSAMLProviderCommand({
    Name: providerName,
    SAMLMetadataDocument: sampleMetadataDocument.toString(),
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};

```

- Einzelheiten zur API finden Sie unter SAMLProvider In der AWS SDK für JavaScript API-Referenz [erstellen](#).

CreateServiceLinkedRole

Das folgende Codebeispiel zeigt die Verwendung `CreateServiceLinkedRole`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen einer serviceverknüpften Rolle.

```
import {
  CreateServiceLinkedRoleCommand,
  GetRoleCommand,
  IAMClient,
} from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} serviceName
 */
export const createServiceLinkedRole = async (serviceName) => {
  const command = new CreateServiceLinkedRoleCommand({
    // For a list of AWS services that support service-linked roles,
    // see https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_aws-services-
    // that-work-with-iam.html.
    //
    // For a list of AWS service endpoints, see https://docs.aws.amazon.com/general/
    // latest/gr/aws-service-information.html.
    AWSServiceName: serviceName,
  });
  try {
    const response = await client.send(command);
    console.log(response);
    return response;
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "InvalidInputException" &&
      caught.message.includes(
        "Service role name AWSServiceRoleForElasticBeanstalk has been taken in this
account",
      )
    ) {
      console.warn(caught.message);
      return client.send(
        new GetRoleCommand({ RoleName: "AWSServiceRoleForElasticBeanstalk" }),
      );
    }
    throw caught;
  }
}
```

```
};
```

- Einzelheiten zur API finden Sie [CreateServiceLinkedRole](#) in der AWS SDK für JavaScript API-Referenz.

CreateUser

Das folgende Codebeispiel zeigt die Verwendung `CreateUser`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie den Benutzer.

```
import { CreateUserCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} name
 */
export const createUser = (name) => {
  const command = new CreateUserCommand({ UserName: name });
  return client.send(command);
};
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [CreateUser](#) in der AWS SDK für JavaScript API-Referenz.

DeleteAccessKey

Das folgende Codebeispiel zeigt die Verwendung `DeleteAccessKey`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Löschen Sie den Zugriffsschlüssel.

```
import { DeleteAccessKeyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} userName
 * @param {string} accessKeyId
 */
export const deleteAccessKey = (userName, accessKeyId) => {
  const command = new DeleteAccessKeyCommand({
    AccessKeyId: accessKeyId,
    UserName: userName,
  });

  return client.send(command);
};
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [DeleteAccessKey](#) in der AWS SDK für JavaScript API-Referenz.

DeleteAccountAlias

Das folgende Codebeispiel zeigt die Verwendung `DeleteAccountAlias`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Löschen Sie den Konto-Alias.

```
import { DeleteAccountAliasCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} alias
 */
export const deleteAccountAlias = (alias) => {
  const command = new DeleteAccountAliasCommand({ AccountAlias: alias });

  return client.send(command);
};
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [DeleteAccountAlias](#) in der AWS SDK für JavaScript API-Referenz.

DeleteGroup

Das folgende Codebeispiel zeigt die Verwendung `DeleteGroup`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { DeleteGroupCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} groupName
 */
export const deleteGroup = async (groupName) => {
  const command = new DeleteGroupCommand({
    GroupName: groupName,
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Einzelheiten zur API finden Sie [DeleteGroup](#) in der AWS SDK für JavaScript API-Referenz.

DeleteInstanceProfile

Das folgende Codebeispiel zeigt die Verwendung `DeleteInstanceProfile`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
const client = new IAMClient({});
await client.send(
  new DeleteInstanceProfileCommand({
    InstanceProfileName: NAMES.instanceProfileName,
  }),
);
```


- Einzelheiten zur API finden Sie [DeleteInstanceProfile](#) in der AWS SDK für JavaScript API-Referenz.

DeletePolicy

Das folgende Codebeispiel zeigt die Verwendung `DeletePolicy`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Löschen Sie die Richtlinie.

```
import { DeletePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} policyArn
 */
export const deletePolicy = (policyArn) => {
  const command = new DeletePolicyCommand({ PolicyArn: policyArn });
  return client.send(command);
};
```

- Einzelheiten zur API finden Sie [DeletePolicy](#) in der AWS SDK für JavaScript API-Referenz.

DeleteRole

Das folgende Codebeispiel zeigt die Verwendung `DeleteRole`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Löschen Sie die Rolle.

```
import { DeleteRoleCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} roleName
 */
export const deleteRole = (roleName) => {
  const command = new DeleteRoleCommand({ RoleName: roleName });
  return client.send(command);
};
```

- Einzelheiten zur API finden Sie [DeleteRole](#) in der AWS SDK für JavaScript API-Referenz.

DeleteRolePolicy

Das folgende Codebeispiel zeigt die Verwendung `DeleteRolePolicy`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { DeleteRolePolicyCommand, IAMClient } from "@aws-sdk/client-iam";
```

```
const client = new IAMClient({});

/**
 *
 * @param {string} roleName
 * @param {string} policyName
 */
export const deleteRolePolicy = (roleName, policyName) => {
  const command = new DeleteRolePolicyCommand({
    RoleName: roleName,
    PolicyName: policyName,
  });
  return client.send(command);
};
```

- Einzelheiten zur API finden Sie [DeleteRolePolicy](#) in der AWS SDK für JavaScript API-Referenz.

DeleteSAMLProvider

Das folgende Codebeispiel zeigt die Verwendung `DeleteSAMLProvider`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { DeleteSAMLProviderCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} providerArn
 * @returns
 */
export const deleteSAMLProvider = async (providerArn) => {
```

```
const command = new DeleteSAMLProviderCommand({
  SAMLProviderArn: providerArn,
});

const response = await client.send(command);
console.log(response);
return response;
};
```

- Einzelheiten zur API finden Sie unter [Löschen SAMLProvider](#) in der AWS SDK für JavaScript API-Referenz.

DeleteServerCertificate

Das folgende Codebeispiel zeigt die Verwendung `DeleteServerCertificate`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Löschen Sie ein Serverzertifikat.

```
import { DeleteServerCertificateCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} certName
 */
export const deleteServerCertificate = (certName) => {
  const command = new DeleteServerCertificateCommand({
    ServerCertificateName: certName,
  });

  return client.send(command);
};
```

```
};
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [DeleteServerCertificate](#) in der AWS SDK für JavaScript API-Referenz.

DeleteServiceLinkedRole

Das folgende Codebeispiel zeigt die Verwendung `DeleteServiceLinkedRole`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { DeleteServiceLinkedRoleCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} roleName
 */
export const deleteServiceLinkedRole = (roleName) => {
  const command = new DeleteServiceLinkedRoleCommand({ RoleName: roleName });
  return client.send(command);
};
```

- Einzelheiten zur API finden Sie [DeleteServiceLinkedRole](#) in der AWS SDK für JavaScript API-Referenz.

DeleteUser

Das folgende Codebeispiel zeigt die Verwendung `DeleteUser`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Löschen Sie den Benutzer.

```
import { DeleteUserCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} name
 */
export const deleteUser = (name) => {
  const command = new DeleteUserCommand({ UserName: name });
  return client.send(command);
};
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [DeleteUser](#) in der AWS SDK für JavaScript API-Referenz.

DetachRolePolicy

Das folgende Codebeispiel zeigt die Verwendung `DetachRolePolicy`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Trennen Sie die Richtlinie.

```
import { DetachRolePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} policyArn
 * @param {string} roleName
 */
export const detachRolePolicy = (policyArn, roleName) => {
  const command = new DetachRolePolicyCommand({
    PolicyArn: policyArn,
    RoleName: roleName,
  });

  return client.send(command);
};
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [DetachRolePolicy](#) in der AWS SDK für JavaScript API-Referenz.

GetAccessKeyLastUsed

Das folgende Codebeispiel zeigt die Verwendung `GetAccessKeyLastUsed`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Rufen Sie den Zugriffsschlüssel ab.

```
import { GetAccessKeyLastUsedCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
```

```
*
* @param {string} accessKeyId
*/
export const getAccessKeyLastUsed = async (accessKeyId) => {
  const command = new GetAccessKeyLastUsedCommand({
    AccessKeyId: accessKeyId,
  });

  const response = await client.send(command);

  if (response.AccessKeyLastUsed?.LastUsedDate) {
    console.log(`
    ${accessKeyId} was last used by ${response.UserName} via
    the ${response.AccessKeyLastUsed.ServiceName} service on
    ${response.AccessKeyLastUsed.LastUsedDate.toISOString()}
    `);
  }

  return response;
};
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [GetAccessKeyLastUsed](#) in der AWS SDK für JavaScript API-Referenz.

GetAccountPasswordPolicy

Das folgende Codebeispiel zeigt die Verwendung `GetAccountPasswordPolicy`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Rufen Sie die Passwort-Richtlinie des Kontos ab.

```
import {
```



```

    GetAccountPasswordPolicyCommand,
    IAMClient,
  } from "@aws-sdk/client-iam";

  const client = new IAMClient({});

  export const getAccountPasswordPolicy = async () => {
    const command = new GetAccountPasswordPolicyCommand({});

    const response = await client.send(command);
    console.log(response.PasswordPolicy);
    return response;
  };

```

- Einzelheiten zur API finden Sie [GetAccountPasswordPolicy](#) in der AWS SDK für JavaScript API-Referenz.

GetPolicy

Das folgende Codebeispiel zeigt die Verwendung `GetPolicy`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Rufen Sie die Richtlinie ab.

```

import { GetPolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} policyArn
 */
export const getPolicy = (policyArn) => {

```

```
const command = new GetPolicyCommand({
  PolicyArn: policyArn,
});

return client.send(command);
};
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [GetPolicy](#) in der AWS SDK für JavaScript API-Referenz.

GetRole

Das folgende Codebeispiel zeigt die Verwendung `GetRole`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Rufen Sie die Rolle ab.

```
import { GetRoleCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} roleName
 */
export const getRole = (roleName) => {
  const command = new GetRoleCommand({
    RoleName: roleName,
  });

  return client.send(command);
};
```

- Einzelheiten zur API finden Sie [GetRole](#) in der AWS SDK für JavaScript API-Referenz.

GetServerCertificate

Das folgende Codebeispiel zeigt die Verwendung `GetServerCertificate`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Rufen Sie ein Serverzertifikat ab.

```
import { GetServerCertificateCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} certName
 * @returns
 */
export const getServerCertificate = async (certName) => {
  const command = new GetServerCertificateCommand({
    ServerCertificateName: certName,
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [GetServerCertificate](#) in der AWS SDK für JavaScript API-Referenz.

GetServiceLinkedRoleDeletionStatus

Das folgende Codebeispiel zeigt die Verwendung `GetServiceLinkedRoleDeletionStatus`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import {
  GetServiceLinkedRoleDeletionStatusCommand,
  IAMClient,
} from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} deletionTaskId
 */
export const getServiceLinkedRoleDeletionStatus = (deletionTaskId) => {
  const command = new GetServiceLinkedRoleDeletionStatusCommand({
    DeletionTaskId: deletionTaskId,
  });

  return client.send(command);
};
```

- Einzelheiten zur API finden Sie [GetServiceLinkedRoleDeletionStatus](#) in der AWS SDK für JavaScript API-Referenz.

ListAccessKeys

Das folgende Codebeispiel zeigt die Verwendung `ListAccessKeys`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Listen Sie die Zugriffsschlüssel auf.

```
import { ListAccessKeysCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/
 * AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify
 * this.
 *
 * @param {string} userName
 */
export async function* listAccessKeys(userName) {
  const command = new ListAccessKeysCommand({
    MaxItems: 5,
    UserName: userName,
  });

  /**
   * @type {import("@aws-sdk/client-iam").ListAccessKeysCommandOutput | undefined}
   */
  let response = await client.send(command);

  while (response?.AccessKeyMetadata?.length) {
    for (const key of response.AccessKeyMetadata) {
      yield key;
    }

    if (response.IsTruncated) {
      response = await client.send(
        new ListAccessKeysCommand({
          Marker: response.Marker,
        }),
      );
    }
  }
}
```

```
    );  
  } else {  
    break;  
  }  
}  
}
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [ListAccessKeys](#) in der AWS SDK für JavaScript API-Referenz.

ListAccountAliases

Das folgende Codebeispiel zeigt die Verwendung `ListAccountAliases`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Listen Sie die Konto-Aliase auf.

```
import { ListAccountAliasesCommand, IAMClient } from "@aws-sdk/client-iam";  
  
const client = new IAMClient({});  
  
/**  
 * A generator function that handles paginated results.  
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/  
AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify  
 this.  
 */  
export async function* listAccountAliases() {  
  const command = new ListAccountAliasesCommand({ MaxItems: 5 });  
  
  let response = await client.send(command);  
  
  while (response.AccountAliases?.length) {  
    for (const alias of response.AccountAliases) {
```

```
    yield alias;
  }

  if (response.IsTruncated) {
    response = await client.send(
      new ListAccountAliasesCommand({
        Marker: response.Marker,
        MaxItems: 5,
      }),
    );
  } else {
    break;
  }
}
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [ListAccountAliases](#) in der AWS SDK für JavaScript API-Referenz.

ListAttachedRolePolicies

Das folgende Codebeispiel zeigt die Verwendung `ListAttachedRolePolicies`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Listen Sie die Richtlinien auf, die an eine Rolle angefügt sind.

```
import {
  ListAttachedRolePoliciesCommand,
  IAMClient,
} from "@aws-sdk/client-iam";

const client = new IAMClient({});
```

```
/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/
AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify
this.
 * @param {string} roleName
 */
export async function* listAttachedRolePolicies(roleName) {
  const command = new ListAttachedRolePoliciesCommand({
    RoleName: roleName,
  });

  let response = await client.send(command);

  while (response.AttachedPolicies?.length) {
    for (const policy of response.AttachedPolicies) {
      yield policy;
    }

    if (response.IsTruncated) {
      response = await client.send(
        new ListAttachedRolePoliciesCommand({
          RoleName: roleName,
          Marker: response.Marker,
        }),
      );
    } else {
      break;
    }
  }
}
```

- Einzelheiten zur API finden Sie [ListAttachedRolePolicies](#) in der AWS SDK für JavaScript API-Referenz.

ListGroups

Das folgende Codebeispiel zeigt die Verwendung `ListGroups`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Listen Sie die Gruppen auf.

```
import { ListGroupsCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/
 * AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify
 * this.
 */
export async function* listGroups() {
  const command = new ListGroupsCommand({
    MaxItems: 10,
  });

  let response = await client.send(command);

  while (response.Groups?.length) {
    for (const group of response.Groups) {
      yield group;
    }

    if (response.IsTruncated) {
      response = await client.send(
        new ListGroupsCommand({
          Marker: response.Marker,
          MaxItems: 10,
        }),
      );
    } else {
      break;
    }
  }
}
```

```
}
```

- Einzelheiten zur API finden Sie [ListGroups](#) in der AWS SDK für JavaScript API-Referenz.

ListPolicies

Das folgende Codebeispiel zeigt die Verwendung `ListPolicies`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Listet Sie die Richtlinien auf.

```
import { ListPoliciesCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/
 * AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify
 * this.
 *
 */
export async function* listPolicies() {
  const command = new ListPoliciesCommand({
    MaxItems: 10,
    OnlyAttached: false,
    // List only the customer managed policies in your Amazon Web Services account.
    Scope: "Local",
  });

  let response = await client.send(command);

  while (response.Policies?.length) {
    for (const policy of response.Policies) {
```

```
    yield policy;
  }

  if (response.IsTruncated) {
    response = await client.send(
      new ListPoliciesCommand({
        Marker: response.Marker,
        MaxItems: 10,
        OnlyAttached: false,
        Scope: "Local",
      }),
    );
  } else {
    break;
  }
}
```

- Einzelheiten zur API finden Sie [ListPolicies](#) in der AWS SDK für JavaScript API-Referenz.

ListRolePolicies

Das folgende Codebeispiel zeigt die Verwendung `ListRolePolicies`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Listet Sie die Richtlinien auf.

```
import { ListRolePoliciesCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
```

```
* The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/
AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify
this.
*
* @param {string} roleName
*/
export async function* listRolePolicies(roleName) {
  const command = new ListRolePoliciesCommand({
    RoleName: roleName,
    MaxItems: 10,
  });

  let response = await client.send(command);

  while (response.PolicyNames?.length) {
    for (const policyName of response.PolicyNames) {
      yield policyName;
    }

    if (response.IsTruncated) {
      response = await client.send(
        new ListRolePoliciesCommand({
          RoleName: roleName,
          MaxItems: 10,
          Marker: response.Marker,
        }),
      );
    } else {
      break;
    }
  }
}
```

- Einzelheiten zur API finden Sie [ListRolePolicies](#) in der AWS SDK für JavaScript API-Referenz.

ListRoles

Das folgende Codebeispiel zeigt die Verwendung `ListRoles`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Listen Sie die Rollen auf.

```
import { ListRolesCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/
 * AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify
 * this.
 */
export async function* listRoles() {
  const command = new ListRolesCommand({
    MaxItems: 10,
  });

  /**
   * @type {import("@aws-sdk/client-iam").ListRolesCommandOutput | undefined}
   */
  let response = await client.send(command);

  while (response?.Roles?.length) {
    for (const role of response.Roles) {
      yield role;
    }

    if (response.IsTruncated) {
      response = await client.send(
        new ListRolesCommand({
          Marker: response.Marker,
        }),
      );
    } else {
```

```
        break;
    }
}
}
```

- Einzelheiten zur API finden Sie [ListRoles](#) in der AWS SDK für JavaScript API-Referenz.

ListSAMLProviders

Das folgende Codebeispiel zeigt die Verwendung `ListSAMLProviders`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Listen Sie die SAML-Anbieter auf.

```
import { ListSAMLProvidersCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

export const listSamlProviders = async () => {
  const command = new ListSAMLProvidersCommand({});

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Einzelheiten zur API finden Sie unter [Liste SAMLProviders](#) in der AWS SDK für JavaScript API-Referenz.

ListServerCertificates

Das folgende Codebeispiel zeigt die Verwendung `ListServerCertificates`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Listen Sie die Zertifikate auf.

```
import { ListServerCertificatesCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 * A generator function that handles paginated results.
 * The AWS SDK for JavaScript (v3) provides {@link https://docs.aws.amazon.com/
 * AWSJavaScriptSDK/v3/latest/index.html#paginators | paginator} functions to simplify
 * this.
 */
export async function* listServerCertificates() {
  const command = new ListServerCertificatesCommand({});
  let response = await client.send(command);

  while (response.ServerCertificateMetadataList?.length) {
    for await (const cert of response.ServerCertificateMetadataList) {
      yield cert;
    }

    if (response.IsTruncated) {
      response = await client.send(new ListServerCertificatesCommand({}));
    } else {
      break;
    }
  }
}
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [ListServerCertificates](#) in der AWS SDK für JavaScript API-Referenz.

ListUsers

Das folgende Codebeispiel zeigt die Verwendung `ListUsers`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Listen Sie die Benutzer auf.

```
import { ListUsersCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

export const listUsers = async () => {
  const command = new ListUsersCommand({ MaxItems: 10 });

  const response = await client.send(command);

  for (const { UserName, CreateDate } of response.Users) {
    console.log(`${UserName} created on: ${CreateDate}`);
  }
  return response;
};
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [ListUsers](#) in der AWS SDK für JavaScript API-Referenz.

PutRolePolicy

Das folgende Codebeispiel zeigt die Verwendung `PutRolePolicy`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { PutRolePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const examplePolicyDocument = JSON.stringify({
  Version: "2012-10-17",
  Statement: [
    {
      Sid: "VisualEditor0",
      Effect: "Allow",
      Action: [
        "s3:ListBucketMultipartUploads",
        "s3:ListBucketVersions",
        "s3:ListBucket",
        "s3:ListMultipartUploadParts",
      ],
      Resource: "arn:aws:s3:::amzn-s3-demo-bucket",
    },
    {
      Sid: "VisualEditor1",
      Effect: "Allow",
      Action: [
        "s3:ListStorageLensConfigurations",
        "s3:ListAccessPointsForObjectLambda",
        "s3:ListAllMyBuckets",
        "s3:ListAccessPoints",
        "s3:ListJobs",
        "s3:ListMultiRegionAccessPoints",
      ],
      Resource: "*",
    },
  ],
});

const client = new IAMClient({});
```

```
/**
 *
 * @param {string} roleName
 * @param {string} policyName
 * @param {string} policyDocument
 */
export const putRolePolicy = async (roleName, policyName, policyDocument) => {
  const command = new PutRolePolicyCommand({
    RoleName: roleName,
    PolicyName: policyName,
    PolicyDocument: policyDocument,
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Einzelheiten zur API finden Sie [PutRolePolicy](#) in der AWS SDK für JavaScript API-Referenz.

UpdateAccessKey

Das folgende Codebeispiel zeigt die Verwendung `UpdateAccessKey`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Aktualisieren Sie den Zugriffsschlüssel.

```
import {
  UpdateAccessKeyCommand,
  IAMClient,
  StatusType,
} from "@aws-sdk/client-iam";

const client = new IAMClient({});
```

```
/**
 *
 * @param {string} userName
 * @param {string} accessKeyId
 */
export const updateAccessKey = (userName, accessKeyId) => {
  const command = new UpdateAccessKeyCommand({
    AccessKeyId: accessKeyId,
    Status: StatusType.Inactive,
    UserName: userName,
  });

  return client.send(command);
};
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [UpdateAccessKey](#) in der AWS SDK für JavaScript API-Referenz.

UpdateServerCertificate

Das folgende Codebeispiel zeigt die Verwendung `UpdateServerCertificate`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Aktualisieren Sie ein Serverzertifikat.

```
import { UpdateServerCertificateCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} currentName
 * @param {string} newName
```

```
*/  
export const updateServerCertificate = (currentName, newName) => {  
  const command = new UpdateServerCertificateCommand({  
    ServerCertificateName: currentName,  
    NewServerCertificateName: newName,  
  });  
  
  return client.send(command);  
};
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [UpdateServerCertificate](#) in der AWS SDK für JavaScript API-Referenz.

UpdateUser

Das folgende Codebeispiel zeigt die Verwendung `UpdateUser`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Aktualisieren Sie den Benutzer.

```
import { UpdateUserCommand, IAMClient } from "@aws-sdk/client-iam";  
  
const client = new IAMClient({});  
  
/**  
 *  
 * @param {string} currentUserName  
 * @param {string} newUserName  
 */  
export const updateUser = (currentUserName, newUserName) => {  
  const command = new UpdateUserCommand({  
    UserName: currentUserName,
```

```
    NewUserName: newUserName,
  });

  return client.send(command);
};
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [UpdateUser](#) in der AWS SDK für JavaScript API-Referenz.

UploadServerCertificate

Das folgende Codebeispiel zeigt die Verwendung `UploadServerCertificate`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { UploadServerCertificateCommand, IAMClient } from "@aws-sdk/client-iam";
import { readFileSync } from "node:fs";
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import * as path from "node:path";

const client = new IAMClient({});

const certMessage = `Generate a certificate and key with the following command, or
the equivalent for your system.

openssl req -x509 -newkey rsa:4096 -sha256 -days 3650 -nodes \
-keyout example.key -out example.crt -subj "/CN=example.com" \
-addext "subjectAltName=DNS:example.com,DNS:www.example.net,IP:10.0.0.1"
`;

const getCertAndKey = () => {
  try {
    const cert = readFileSync(
      path.join(dirnameFromMetaUrl(import.meta.url), "./example.crt"),
```

```

    );
    const key = readFileSync(
      path.join(dirnameFromMetaUrl(import.meta.url), "./example.key"),
    );
    return { cert, key };
  } catch (err) {
    if (err.code === "ENOENT") {
      throw new Error(
        `Certificate and/or private key not found. ${certMessage}`,
      );
    }
  }

  throw err;
}
};

/**
 *
 * @param {string} certificateName
 */
export const uploadServerCertificate = (certificateName) => {
  const { cert, key } = getCertAndKey();
  const command = new UploadServerCertificateCommand({
    ServerCertificateName: certificateName,
    CertificateBody: cert.toString(),
    PrivateKey: key.toString(),
  });

  return client.send(command);
};

```

- Einzelheiten zur API finden Sie [UploadServerCertificate](#) in der AWS SDK für JavaScript API-Referenz.

Szenarien

Erstellen und Verwalten eines ausfallsicheren Services

Das folgende Codebeispiel zeigt, wie Sie einen Webservice mit Load Balancing erstellen, der Buch-, Film- und Liedempfehlungen zurückgibt. Das Beispiel zeigt, wie der Service auf Fehler reagiert und wie der Service für mehr Ausfallsicherheit umstrukturiert werden kann.

- Verwenden Sie eine Amazon EC2 Auto Scaling Scaling-Gruppe, um Amazon Elastic Compute Cloud (Amazon EC2) -Instances auf der Grundlage einer Startvorlage zu erstellen und die Anzahl der Instances in einem bestimmten Bereich zu halten.
- Verarbeiten und verteilen Sie HTTP-Anfragen mit Elastic Load Balancing.
- Überwachen Sie den Zustand von Instances in einer Auto-Scaling-Gruppe und leiten Sie Anfragen nur an fehlerfreie Instances weiter.
- Führen Sie auf jeder EC2 Instanz einen Python-Webserver aus, um HTTP-Anfragen zu verarbeiten. Der Webserver reagiert mit Empfehlungen und Zustandsprüfungen.
- Simulieren Sie einen Empfehlungsservice mit einer Amazon DynamoDB-Tabelle.
- Steuern Sie die Antwort des Webserver auf Anfragen und Zustandsprüfungen, indem Sie die AWS Systems Manager Parameter aktualisieren.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Führen Sie ein interaktives Szenario an einer Eingabeaufforderung aus.

```
#!/usr/bin/env node
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0

import {
  Scenario,
  parseScenarioArgs,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

/**
 * The workflow steps are split into three stages:
 *   - deploy
 *   - demo
 *   - destroy
 *
 * Each of these stages has a corresponding file prefixed with steps-*.
 */
```

```

import { deploySteps } from "./steps-deploy.js";
import { demoSteps } from "./steps-demo.js";
import { destroySteps } from "./steps-destroy.js";

/**
 * The context is passed to every scenario. Scenario steps
 * will modify the context.
 */
const context = {};

/**
 * Three Scenarios are created for the workflow. A Scenario is an orchestration
 * class
 * that simplifies running a series of steps.
 */
export const scenarios = {
  // Deploys all resources necessary for the workflow.
  deploy: new Scenario("Resilient Workflow - Deploy", deploySteps, context),
  // Demonstrates how a fragile web service can be made more resilient.
  demo: new Scenario("Resilient Workflow - Demo", demoSteps, context),
  // Destroys the resources created for the workflow.
  destroy: new Scenario("Resilient Workflow - Destroy", destroySteps, context),
};

// Call function if run directly
import { fileURLToPath } from "node:url";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios, {
    name: "Resilient Workflow",
    synopsis:
      "node index.js --scenario <deploy | demo | destroy> [-h|--help] [-y|--yes] [-v|--verbose]",
    description: "Deploy and interact with scalable EC2 instances.",
  });
}

```

Erstellen Sie Schritte, um alle Ressourcen bereitzustellen.

```

import { join } from "node:path";
import { readFileSync, writeFileSync } from "node:fs";
import axios from "axios";

```



```
import {
  BatchWriteItemCommand,
  CreateTableCommand,
  DynamoDBClient,
  waitUntilTableExists,
} from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  CreateKeyPairCommand,
  CreateLaunchTemplateCommand,
  DescribeAvailabilityZonesCommand,
  DescribeVpcsCommand,
  DescribeSubnetsCommand,
  DescribeSecurityGroupsCommand,
  AuthorizeSecurityGroupIngressCommand,
} from "@aws-sdk/client-ec2";
import {
  IAMClient,
  CreatePolicyCommand,
  CreateRoleCommand,
  CreateInstanceProfileCommand,
  AddRoleToInstanceProfileCommand,
  AttachRolePolicyCommand,
  waitUntilInstanceProfileExists,
} from "@aws-sdk/client-iam";
import { SSMClient, GetParameterCommand } from "@aws-sdk/client-ssm";
import {
  CreateAutoScalingGroupCommand,
  AutoScalingClient,
  AttachLoadBalancerTargetGroupsCommand,
} from "@aws-sdk/client-auto-scaling";
import {
  CreateListenerCommand,
  CreateLoadBalancerCommand,
  CreateTargetGroupCommand,
  ElasticLoadBalancingV2Client,
  waitUntilLoadBalancerAvailable,
} from "@aws-sdk/client-elastic-load-balancing-v2";

import {
  ScenarioOutput,
  ScenarioInput,
  ScenarioAction,
```

```

} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { saveState } from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES, RESOURCES_PATH, ROOT } from "../constants.js";
import { initParamsSteps } from "../steps-reset-params.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const deploySteps = [
  new ScenarioOutput("introduction", MESSAGES.introduction, { header: true }),
  new ScenarioInput("confirmDeployment", MESSAGES.confirmDeployment, {
    type: "confirm",
  }),
  new ScenarioAction(
    "handleConfirmDeployment",
    (c) => c.confirmDeployment === false && process.exit(),
  ),
  new ScenarioOutput(
    "creatingTable",
    MESSAGES.creatingTable.replace("${TABLE_NAME}", NAMES.tableName),
  ),
  new ScenarioAction("createTable", async () => {
    const client = new DynamoDBClient({});
    await client.send(
      new CreateTableCommand({
        TableName: NAMES.tableName,
        ProvisionedThroughput: {
          ReadCapacityUnits: 5,
          WriteCapacityUnits: 5,
        },
        AttributeDefinitions: [
          {
            AttributeName: "MediaType",
            AttributeType: "S",
          },
          {
            AttributeName: "ItemId",
            AttributeType: "N",
          },
        ],
        KeySchema: [
          {

```

```

        AttributeName: "MediaType",
        KeyType: "HASH",
      },
      {
        AttributeName: "ItemId",
        KeyType: "RANGE",
      },
    ],
  )),
);
await waitUntilTableExists({ client }, { TableName: NAMES.tableName });
}),
new ScenarioOutput(
  "createdTable",
  MESSAGES.createdTable.replace("${TABLE_NAME}", NAMES.tableName),
),
new ScenarioOutput(
  "populatingTable",
  MESSAGES.populatingTable.replace("${TABLE_NAME}", NAMES.tableName),
),
new ScenarioAction("populateTable", () => {
  const client = new DynamoDBClient({});
  /**
   * @type {{ default: import("@aws-sdk/client-dynamodb").PutRequest['Item'][] }}
   */
  const recommendations = JSON.parse(
    readFileSync(join(RESOURCES_PATH, "recommendations.json")),
  );

  return client.send(
    new BatchWriteItemCommand({
      RequestItems: {
        [NAMES.tableName]: recommendations.map((item) => ({
          PutRequest: { Item: item },
        })),
      },
    }),
  );
}),
new ScenarioOutput(
  "populatedTable",
  MESSAGES.populatedTable.replace("${TABLE_NAME}", NAMES.tableName),
),
new ScenarioOutput(

```

```

    "creatingKeyPair",
    MESSAGES.creatingKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
  ),
  new ScenarioAction("createKeyPair", async () => {
    const client = new EC2Client({});
    const { KeyMaterial } = await client.send(
      new CreateKeyPairCommand({
        KeyName: NAMES.keyPairName,
      }),
    );

    writeFileSync(`${NAMES.keyPairName}.pem`, KeyMaterial, { mode: 0o600 });
  }),
  new ScenarioOutput(
    "createdKeyPair",
    MESSAGES.createdKeyPair.replace("${KEY_PAIR_NAME}", NAMES.keyPairName),
  ),
  new ScenarioOutput(
    "creatingInstancePolicy",
    MESSAGES.creatingInstancePolicy.replace(
      "${INSTANCE_POLICY_NAME}",
      NAMES.instancePolicyName,
    ),
  ),
  new ScenarioAction("createInstancePolicy", async (state) => {
    const client = new IAMClient({});
    const {
      Policy: { Arn },
    } = await client.send(
      new CreatePolicyCommand({
        PolicyName: NAMES.instancePolicyName,
        PolicyDocument: readFileSync(
          join(RESOURCES_PATH, "instance_policy.json"),
        ),
      }),
    );
    state.instancePolicyArn = Arn;
  }),
  new ScenarioOutput("createdInstancePolicy", (state) =>
    MESSAGES.createdInstancePolicy
      .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
      .replace("${INSTANCE_POLICY_ARN}", state.instancePolicyArn),
  ),
  new ScenarioOutput(

```

```

    "creatingInstanceRole",
    MESSAGES.creatingInstanceRole.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
    ),
),
new ScenarioAction("createInstanceRole", () => {
    const client = new IAMClient({});
    return client.send(
        new CreateRoleCommand({
            RoleName: NAMES.instanceRoleName,
            AssumeRolePolicyDocument: readFileSync(
                join(ROOT, "assume-role-policy.json"),
            ),
        }),
    );
}),
new ScenarioOutput(
    "createdInstanceRole",
    MESSAGES.createdInstanceRole.replace(
        "${INSTANCE_ROLE_NAME}",
        NAMES.instanceRoleName,
    ),
),
new ScenarioOutput(
    "attachingPolicyToRole",
    MESSAGES.attachingPolicyToRole
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName)
        .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName),
),
new ScenarioAction("attachPolicyToRole", async (state) => {
    const client = new IAMClient({});
    await client.send(
        new AttachRolePolicyCommand({
            RoleName: NAMES.instanceRoleName,
            PolicyArn: state.instancePolicyArn,
        }),
    );
}),
new ScenarioOutput(
    "attachedPolicyToRole",
    MESSAGES.attachedPolicyToRole
        .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),

```

```
),
new ScenarioOutput(
  "creatingInstanceProfile",
  MESSAGES.creatingInstanceProfile.replace(
    "${INSTANCE_PROFILE_NAME}",
    NAMES.instanceProfileName,
  ),
),
new ScenarioAction("createInstanceProfile", async (state) => {
  const client = new IAMClient({});
  const {
    InstanceProfile: { Arn },
  } = await client.send(
    new CreateInstanceProfileCommand({
      InstanceProfileName: NAMES.instanceProfileName,
    }),
  );
  state.instanceProfileArn = Arn;

  await waitUntilInstanceProfileExists(
    { client },
    { InstanceProfileName: NAMES.instanceProfileName },
  );
}),
new ScenarioOutput("createdInstanceProfile", (state) =>
  MESSAGES.createdInstanceProfile
    .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
    .replace("${INSTANCE_PROFILE_ARN}", state.instanceProfileArn),
),
new ScenarioOutput(
  "addingRoleToInstanceProfile",
  MESSAGES.addingRoleToInstanceProfile
    .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
    .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
),
new ScenarioAction("addRoleToInstanceProfile", () => {
  const client = new IAMClient({});
  return client.send(
    new AddRoleToInstanceProfileCommand({
      RoleName: NAMES.instanceRoleName,
      InstanceProfileName: NAMES.instanceProfileName,
    }),
  );
}),
```

```

new ScenarioOutput(
  "addedRoleToInstanceProfile",
  MESSAGES.addedRoleToInstanceProfile
    .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
    .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName),
),
...initParamsSteps,
new ScenarioOutput("creatingLaunchTemplate", MESSAGES.creatingLaunchTemplate),
new ScenarioAction("createLaunchTemplate", async () => {
  const ssmClient = new SSMClient({});
  const { Parameter } = await ssmClient.send(
    new GetParameterCommand({
      Name: "/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2",
    }),
  );
  const ec2Client = new EC2Client({});
  await ec2Client.send(
    new CreateLaunchTemplateCommand({
      LaunchTemplateName: NAMES.launchTemplateName,
      LaunchTemplateData: {
        InstanceType: "t3.micro",
        ImageId: Parameter.Value,
        IamInstanceProfile: { Name: NAMES.instanceProfileName },
        UserData: readFileSync(
          join(RESOURCES_PATH, "server_startup_script.sh"),
        ).toString("base64"),
        KeyName: NAMES.keyPairName,
      },
    }),
  );
}),
new ScenarioOutput(
  "createdLaunchTemplate",
  MESSAGES.createdLaunchTemplate.replace(
    "${LAUNCH_TEMPLATE_NAME}",
    NAMES.launchTemplateName,
  ),
),
new ScenarioOutput(
  "creatingAutoScalingGroup",
  MESSAGES.creatingAutoScalingGroup.replace(
    "${AUTO_SCALING_GROUP_NAME}",
    NAMES.autoScalingGroupName,
  ),
),

```

```

    ),
    new ScenarioAction("createAutoScalingGroup", async (state) => {
      const ec2Client = new EC2Client({});
      const { AvailabilityZones } = await ec2Client.send(
        new DescribeAvailabilityZonesCommand({}),
      );
      state.availabilityZoneNames = AvailabilityZones.map((az) => az.ZoneName);
      const autoScalingClient = new AutoScalingClient({});
      await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
        autoScalingClient.send(
          new CreateAutoScalingGroupCommand({
            AvailabilityZones: state.availabilityZoneNames,
            AutoScalingGroupName: NAMES.autoScalingGroupName,
            LaunchTemplate: {
              LaunchTemplateName: NAMES.launchTemplateName,
              Version: "$Default",
            },
            MinSize: 3,
            MaxSize: 3,
          }),
        ),
      );
    }),
    new ScenarioOutput(
      "createdAutoScalingGroup",
      /**
       * @param {{ availabilityZoneNames: string[] }} state
       */
      (state) =>
        MESSAGES.createdAutoScalingGroup
          .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName)
          .replace(
            "${AVAILABILITY_ZONE_NAMES}",
            state.availabilityZoneNames.join(", "),
          ),
    ),
    new ScenarioInput("confirmContinue", MESSAGES.confirmContinue, {
      type: "confirm",
    }),
    new ScenarioOutput("loadBalancer", MESSAGES.loadBalancer),
    new ScenarioOutput("gettingVpc", MESSAGES.gettingVpc),
    new ScenarioAction("getVpc", async (state) => {
      const client = new EC2Client({});
      const { Vpcs } = await client.send(

```



```

    new DescribeVpcsCommand({
      Filters: [{ Name: "is-default", Values: ["true"] }],
    }),
  );
  state.defaultVpc = Vpcs[0].VpcId;
}),
new ScenarioOutput("gotVpc", (state) =>
  MESSAGES.gotVpc.replace("${VPC_ID}", state.defaultVpc),
),
new ScenarioOutput("gettingSubnets", MESSAGES.gettingSubnets),
new ScenarioAction("getSubnets", async (state) => {
  const client = new EC2Client({});
  const { Subnets } = await client.send(
    new DescribeSubnetsCommand({
      Filters: [
        { Name: "vpc-id", Values: [state.defaultVpc] },
        { Name: "availability-zone", Values: state.availabilityZoneNames },
        { Name: "default-for-az", Values: ["true"] },
      ],
    }),
  );
  state.subnets = Subnets.map((subnet) => subnet.SubnetId);
}),
new ScenarioOutput(
  "gotSubnets",
  /**
   * @param {{ subnets: string[] }} state
   */
  (state) =>
    MESSAGES.gotSubnets.replace("${SUBNETS}", state.subnets.join(", ")),
),
new ScenarioOutput(
  "creatingLoadBalancerTargetGroup",
  MESSAGES.creatingLoadBalancerTargetGroup.replace(
    "${TARGET_GROUP_NAME}",
    NAMES.loadBalancerTargetGroupName,
  ),
),
new ScenarioAction("createLoadBalancerTargetGroup", async (state) => {
  const client = new ElasticLoadBalancingV2Client({});
  const { TargetGroups } = await client.send(
    new CreateTargetGroupCommand({
      Name: NAMES.loadBalancerTargetGroupName,
      Protocol: "HTTP",
    }),
  );
  state.targetGroup = TargetGroups[0].TargetGroupId;
}),
new ScenarioOutput(
  "creatingLoadBalancer",
  MESSAGES.creatingLoadBalancer.replace(
    "${LOAD_BALANCER_NAME}",
    NAMES.loadBalancerName,
  ),
),
new ScenarioAction("createLoadBalancer", async (state) => {
  const client = new ElasticLoadBalancingV2Client({});
  const { LoadBalancers } = await client.send(
    new CreateLoadBalancerCommand({
      Name: NAMES.loadBalancerName,
      Subnets: state.subnets,
      Schemas: ["ELB_Schema_2015_12_01"],
      LoadBalancerScheme: "internet-facing",
      LoadBalancerType: "application",
    }),
  );
  state.loadBalancer = LoadBalancers[0].LoadBalancerId;
}),
new ScenarioOutput(
  "creatingRouteTable",
  MESSAGES.creatingRouteTable.replace(
    "${ROUTE_TABLE_NAME}",
    NAMES.routeTableName,
  ),
),
new ScenarioAction("createRouteTable", async (state) => {
  const client = new EC2Client({});
  const { RouteTables } = await client.send(
    new CreateRouteTableCommand({
      VpcId: state.defaultVpc,
      SubnetIds: state.subnets,
      RouteTableType: "public",
    }),
  );
  state.routeTable = RouteTables[0].RouteTableId;
}),
new ScenarioOutput(
  "creatingRoute",
  MESSAGES.creatingRoute.replace(
    "${ROUTE_NAME}",
    NAMES.routeName,
  ),
),
new ScenarioAction("createRoute", async (state) => {
  const client = new EC2Client({});
  const { Routes } = await client.send(
    new CreateRouteCommand({
      RouteTableId: state.routeTable,
      DestinationCidrBlock: state.destinationCidrBlock,
      SubnetId: state.subnets[0],
    }),
  );
  state.route = Routes[0].RouteId;
}),
new ScenarioOutput(
  "creatingInternetGateway",
  MESSAGES.creatingInternetGateway.replace(
    "${INTERNET_GATEWAY_NAME}",
    NAMES.internetGatewayName,
  ),
),
new ScenarioAction("createInternetGateway", async (state) => {
  const client = new EC2Client({});
  const { InternetGateways } = await client.send(
    new CreateInternetGatewayCommand({
      InternetGatewayType: "internet",
    }),
  );
  state.internetGateway = InternetGateways[0].InternetGatewayId;
}),
new ScenarioOutput(
  "attachingInternetGateway",
  MESSAGES.attachingInternetGateway.replace(
    "${VPC_ID}",
    state.defaultVpc,
  ),
),
new ScenarioAction("attachInternetGateway", async (state) => {
  const client = new EC2Client({});
  const { Attachments } = await client.send(
    new AttachInternetGatewayCommand({
      InternetGatewayId: state.internetGateway,
      VpcId: state.defaultVpc,
    }),
  );
  state.attachment = Attachments[0].AttachmentId;
}),
new ScenarioOutput(
  "creatingVpcEndpoint",
  MESSAGES.creatingVpcEndpoint.replace(
    "${VPC_ENDPOINT_NAME}",
    NAMES.vpcEndpointName,
  ),
),
new ScenarioAction("createVpcEndpoint", async (state) => {
  const client = new EC2Client({});
  const { VpcEndpoints } = await client.send(
    new CreateVpcEndpointCommand({
      VpcId: state.defaultVpc,
      SubnetIds: state.subnets,
      VpcEndpointType: "interface",
      VpcEndpointName: NAMES.vpcEndpointName,
      VpcEndpointPolicy: "allow-all",
    }),
  );
  state.vpcEndpoint = VpcEndpoints[0].VpcEndpointId;
}),
new ScenarioOutput(
  "creatingLoadBalancer",
  MESSAGES.creatingLoadBalancer.replace(
    "${LOAD_BALANCER_NAME}",
    NAMES.loadBalancerName,
  ),
),
new ScenarioAction("createLoadBalancer", async (state) => {
  const client = new ElasticLoadBalancingV2Client({});
  const { LoadBalancers } = await client.send(
    new CreateLoadBalancerCommand({
      Name: NAMES.loadBalancerName,
      Subnets: state.subnets,
      Schemas: ["ELB_Schema_2015_12_01"],
      LoadBalancerScheme: "internet-facing",
      LoadBalancerType: "application",
    }),
  );
  state.loadBalancer = LoadBalancers[0].LoadBalancerId;
}),
new ScenarioOutput(
  "creatingTargetGroup",
  MESSAGES.creatingTargetGroup.replace(
    "${TARGET_GROUP_NAME}",
    NAMES.loadBalancerTargetGroupName,
  ),
),
new ScenarioAction("createTargetGroup", async (state) => {
  const client = new ElasticLoadBalancingV2Client({});
  const { TargetGroups } = await client.send(
    new CreateTargetGroupCommand({
      Name: NAMES.loadBalancerTargetGroupName,
      Protocol: "HTTP",
    }),
  );
  state.targetGroup = TargetGroups[0].TargetGroupId;
}),
new ScenarioOutput(
  "creatingLoadBalancer",
  MESSAGES.creatingLoadBalancer.replace(
    "${LOAD_BALANCER_NAME}",
    NAMES.loadBalancerName,
  ),
),
new ScenarioAction("createLoadBalancer", async (state) => {
  const client = new ElasticLoadBalancingV2Client({});
  const { LoadBalancers } = await client.send(
    new CreateLoadBalancerCommand({
      Name: NAMES.loadBalancerName,
      Subnets: state.subnets,
      Schemas: ["ELB_Schema_2015_12_01"],
      LoadBalancerScheme: "internet-facing",
      LoadBalancerType: "application",
    }),
  );
  state.loadBalancer = LoadBalancers[0].LoadBalancerId;
}),
new ScenarioOutput(
  "creatingRouteTable",
  MESSAGES.creatingRouteTable.replace(
    "${ROUTE_TABLE_NAME}",
    NAMES.routeTableName,
  ),
),
new ScenarioAction("createRouteTable", async (state) => {
  const client = new EC2Client({});
  const { RouteTables } = await client.send(
    new CreateRouteTableCommand({
      VpcId: state.defaultVpc,
      SubnetIds: state.subnets,
      RouteTableType: "public",
    }),
  );
  state.routeTable = RouteTables[0].RouteTableId;
}),
new ScenarioOutput(
  "creatingRoute",
  MESSAGES.creatingRoute.replace(
    "${ROUTE_NAME}",
    NAMES.routeName,
  ),
),
new ScenarioAction("createRoute", async (state) => {
  const client = new EC2Client({});
  const { Routes } = await client.send(
    new CreateRouteCommand({
      RouteTableId: state.routeTable,
      DestinationCidrBlock: state.destinationCidrBlock,
      SubnetId: state.subnets[0],
    }),
  );
  state.route = Routes[0].RouteId;
}),
new ScenarioOutput(
  "creatingInternetGateway",
  MESSAGES.creatingInternetGateway.replace(
    "${INTERNET_GATEWAY_NAME}",
    NAMES.internetGatewayName,
  ),
),
new ScenarioAction("createInternetGateway", async (state) => {
  const client = new EC2Client({});
  const { InternetGateways } = await client.send(
    new CreateInternetGatewayCommand({
      InternetGatewayType: "internet",
    }),
  );
  state.internetGateway = InternetGateways[0].InternetGatewayId;
}),
new ScenarioOutput(
  "attachingInternetGateway",
  MESSAGES.attachingInternetGateway.replace(
    "${VPC_ID}",
    state.defaultVpc,
  ),
),
new ScenarioAction("attachInternetGateway", async (state) => {
  const client = new EC2Client({});
  const { Attachments } = await client.send(
    new AttachInternetGatewayCommand({
      InternetGatewayId: state.internetGateway,
      VpcId: state.defaultVpc,
    }),
  );
  state.attachment = Attachments[0].AttachmentId;
}),
new ScenarioOutput(
  "creatingVpcEndpoint",
  MESSAGES.creatingVpcEndpoint.replace(
    "${VPC_ENDPOINT_NAME}",
    NAMES.vpcEndpointName,
  ),
),
new ScenarioAction("createVpcEndpoint", async (state) => {
  const client = new EC2Client({});
  const { VpcEndpoints } = await client.send(
    new CreateVpcEndpointCommand({
      VpcId: state.defaultVpc,
      SubnetIds: state.subnets,
      VpcEndpointType: "interface",
      VpcEndpointName: NAMES.vpcEndpointName,
      VpcEndpointPolicy: "allow-all",
    }),
  );
  state.vpcEndpoint = VpcEndpoints[0].VpcEndpointId;
}),

```

```

        Port: 80,
        HealthCheckPath: "/healthcheck",
        HealthCheckIntervalSeconds: 10,
        HealthCheckTimeoutSeconds: 5,
        HealthyThresholdCount: 2,
        UnhealthyThresholdCount: 2,
        VpcId: state.defaultVpc,
    )),
);
const targetGroup = TargetGroups[0];
state.targetGroupArn = targetGroup.TargetGroupArn;
state.targetGroupProtocol = targetGroup.Protocol;
state.targetGroupPort = targetGroup.Port;
}),
new ScenarioOutput(
    "createdLoadBalancerTargetGroup",
    MESSAGES.createdLoadBalancerTargetGroup.replace(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
    ),
),
new ScenarioOutput(
    "creatingLoadBalancer",
    MESSAGES.creatingLoadBalancer.replace("${LB_NAME}", NAMES.loadBalancerName),
),
new ScenarioAction("createLoadBalancer", async (state) => {
    const client = new ElasticLoadBalancingV2Client({});
    const { LoadBalancers } = await client.send(
        new CreateLoadBalancerCommand({
            Name: NAMES.loadBalancerName,
            Subnets: state.subnets,
        }),
    );
    state.loadBalancerDns = LoadBalancers[0].DNSName;
    state.loadBalancerArn = LoadBalancers[0].LoadBalancerArn;
    await waitUntilLoadBalancerAvailable(
        { client },
        { Names: [NAMES.loadBalancerName] },
    );
}),
new ScenarioOutput("createdLoadBalancer", (state) =>
    MESSAGES.createdLoadBalancer
        .replace("${LB_NAME}", NAMES.loadBalancerName)
        .replace("${DNS_NAME}", state.loadBalancerDns),

```

```

    ),
    new ScenarioOutput(
      "creatingListener",
      MESSAGES.creatingLoadBalancerListener
        .replace("${LB_NAME}", NAMES.loadBalancerName)
        .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName),
    ),
    new ScenarioAction("createListener", async (state) => {
      const client = new ElasticLoadBalancingV2Client({});
      const { Listeners } = await client.send(
        new CreateListenerCommand({
          LoadBalancerArn: state.loadBalancerArn,
          Protocol: state.targetGroupProtocol,
          Port: state.targetGroupPort,
          DefaultActions: [
            { Type: "forward", TargetGroupArn: state.targetGroupArn },
          ],
        }),
      );
      const listener = Listeners[0];
      state.loadBalancerListenerArn = listener.ListenerArn;
    }),
    new ScenarioOutput("createdListener", (state) =>
      MESSAGES.createdLoadBalancerListener.replace(
        "${LB_LISTENER_ARN}",
        state.loadBalancerListenerArn,
      ),
    ),
    new ScenarioOutput(
      "attachingLoadBalancerTargetGroup",
      MESSAGES.attachingLoadBalancerTargetGroup
        .replace("${TARGET_GROUP_NAME}", NAMES.loadBalancerTargetGroupName)
        .replace("${AUTO_SCALING_GROUP_NAME}", NAMES.autoScalingGroupName),
    ),
    new ScenarioAction("attachLoadBalancerTargetGroup", async (state) => {
      const client = new AutoScalingClient({});
      await client.send(
        new AttachLoadBalancerTargetGroupsCommand({
          AutoScalingGroupName: NAMES.autoScalingGroupName,
          TargetGroupARNs: [state.targetGroupArn],
        }),
      );
    }),
    new ScenarioOutput(

```

```

    "attachedLoadBalancerTargetGroup",
    MESSAGES.attachedLoadBalancerTargetGroup,
  ),
  new ScenarioOutput("verifyingInboundPort", MESSAGES.verifyingInboundPort),
  new ScenarioAction(
    "verifyInboundPort",
    /**
     *
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-ec2').SecurityGroup}}
state
    */
    async (state) => {
      const client = new EC2Client({});
      const { SecurityGroups } = await client.send(
        new DescribeSecurityGroupsCommand({
          Filters: [{ Name: "group-name", Values: ["default"] }],
        }),
      );
      if (!SecurityGroups) {
        state.verifyInboundPortError = new Error(MESSAGES.noSecurityGroups);
      }
      state.defaultSecurityGroup = SecurityGroups[0];

      /**
       * @type {string}
       */
      const ipResponse = (await axios.get("http://checkip.amazonaws.com")).data;
      state.myIp = ipResponse.trim();
      const myIpRules = state.defaultSecurityGroup.IpPermissions.filter(
        ({ IpRanges }) =>
          IpRanges.some(
            ({ CidrIp }) =>
              CidrIp.startsWith(state.myIp) || CidrIp === "0.0.0.0/0",
          ),
      )
        .filter(({ IpProtocol }) => IpProtocol === "tcp")
        .filter(({ FromPort }) => FromPort === 80);

      state.myIpRules = myIpRules;
    },
  ),
  new ScenarioOutput(
    "verifiedInboundPort",
    /**

```

```

    * @param {{ myIpRules: any[] }} state
    */
    (state) => {
      if (state.myIpRules.length > 0) {
        return MESSAGES.foundIpRules.replace(
          "${IP_RULES}",
          JSON.stringify(state.myIpRules, null, 2),
        );
      }
      return MESSAGES.noIpRules;
    },
  ),
  new ScenarioInput(
    "shouldAddInboundRule",
    /**
     * @param {{ myIpRules: any[] }} state
     */
    (state) => {
      if (state.myIpRules.length > 0) {
        return false;
      }
      return MESSAGES.noIpRules;
    },
    { type: "confirm" },
  ),
  new ScenarioAction(
    "addInboundRule",
    /**
     * @param {{ defaultSecurityGroup: import('@aws-sdk/client-ec2').SecurityGroup }} state
     */
    async (state) => {
      if (!state.shouldAddInboundRule) {
        return;
      }

      const client = new EC2Client({});
      await client.send(
        new AuthorizeSecurityGroupIngressCommand({
          GroupId: state.defaultSecurityGroup.GroupId,
          CidrIp: `${state.myIp}/32`,
          FromPort: 80,
          ToPort: 80,
          IpProtocol: "tcp",

```

```

    }},
  );
},
),
new ScenarioOutput("addedInboundRule", (state) => {
  if (state.shouldAddInboundRule) {
    return MESSAGES.addedInboundRule.replace("${IP_ADDRESS}", state.myIp);
  }
  return false;
}),
new ScenarioOutput("verifyingEndpoint", (state) =>
  MESSAGES.verifyingEndpoint.replace("${DNS_NAME}", state.loadBalancerDns),
),
new ScenarioAction("verifyEndpoint", async (state) => {
  try {
    const response = await retry({ intervalInMs: 2000, maxRetries: 30 }, () =>
      axios.get(`http://${state.loadBalancerDns}`),
    );
    state.endpointResponse = JSON.stringify(response.data, null, 2);
  } catch (e) {
    state.verifyEndpointError = e;
  }
}),
new ScenarioOutput("verifiedEndpoint", (state) => {
  if (state.verifyEndpointError) {
    console.error(state.verifyEndpointError);
  } else {
    return MESSAGES.verifiedEndpoint.replace(
      "${ENDPOINT_RESPONSE}",
      state.endpointResponse,
    );
  }
}),
saveState,
];

```

Erstellen Sie Schritte, um die Demo auszuführen.

```

import { readFileSync } from "node:fs";
import { join } from "node:path";

import axios from "axios";

```

```
import {
  DescribeTargetGroupsCommand,
  DescribeTargetHealthCommand,
  ElasticLoadBalancingV2Client,
} from "@aws-sdk/client-elastic-load-balancing-v2";
import {
  DescribeInstanceInformationCommand,
  PutParameterCommand,
  SSMClient,
  SendCommandCommand,
} from "@aws-sdk/client-ssm";
import {
  IAMClient,
  CreatePolicyCommand,
  CreateRoleCommand,
  AttachRolePolicyCommand,
  CreateInstanceProfileCommand,
  AddRoleToInstanceProfileCommand,
  waitUntilInstanceProfileExists,
} from "@aws-sdk/client-iam";
import {
  AutoScalingClient,
  DescribeAutoScalingGroupsCommand,
  TerminateInstanceInAutoScalingGroupCommand,
} from "@aws-sdk/client-auto-scaling";
import {
  DescribeIamInstanceProfileAssociationsCommand,
  EC2Client,
  RebootInstancesCommand,
  ReplaceIamInstanceProfileAssociationCommand,
} from "@aws-sdk/client-ec2";

import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/scenario.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES, RESOURCES_PATH } from "../constants.js";
import { findLoadBalancer } from "../shared.js";

const getRecommendation = new ScenarioAction(
```

```

    "getRecommendation",
    async (state) => {
      const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
      if (loadBalancer) {
        state.loadBalancerDnsName = loadBalancer.DNSName;
        try {
          state.recommendation = (
            await axios.get(`http://${state.loadBalancerDnsName}`)
          ).data;
        } catch (e) {
          state.recommendation = e instanceof Error ? e.message : e;
        }
      } else {
        throw new Error(MESSAGES.demoFindLoadBalancerError);
      }
    },
  );

  const getRecommendationResult = new ScenarioOutput(
    "getRecommendationResult",
    (state) =>
      `Recommendation:\n${JSON.stringify(state.recommendation, null, 2)}`,
    { preformatted: true },
  );

  const getHealthCheck = new ScenarioAction("getHealthCheck", async (state) => {
    const client = new ElasticLoadBalancingV2Client({});
    const { TargetGroups } = await client.send(
      new DescribeTargetGroupsCommand({
        Names: [NAMES.loadBalancerTargetGroupName],
      }),
    );
  });

  const { TargetHealthDescriptions } = await client.send(
    new DescribeTargetHealthCommand({
      TargetGroupArn: TargetGroups[0].TargetGroupArn,
    }),
  );
  state.targetHealthDescriptions = TargetHealthDescriptions;
});

const getHealthCheckResult = new ScenarioOutput(
  "getHealthCheckResult",
  /**

```



```

    * @param {{ targetHealthDescriptions: import('@aws-sdk/client-elastic-load-
balancing-v2').TargetHealthDescription[]}} state
    */
    (state) => {
      const status = state.targetHealthDescriptions
        .map((th) => `${th.Target.Id}: ${th.TargetHealth.State}`)
        .join("\n");
      return `Health check:\n${status}`;
    },
    { preformatted: true },
  );

const loadBalancerLoop = new ScenarioAction(
  "loadBalancerLoop",
  getRecommendation.action,
  {
    whileConfig: {
      whileFn: ({ loadBalancerCheck }) => loadBalancerCheck,
      input: new ScenarioInput(
        "loadBalancerCheck",
        MESSAGES.demoLoadBalancerCheck,
        {
          type: "confirm",
        },
      ),
      output: getRecommendationResult,
    },
  },
);

const healthCheckLoop = new ScenarioAction(
  "healthCheckLoop",
  getHealthCheck.action,
  {
    whileConfig: {
      whileFn: ({ healthCheck }) => healthCheck,
      input: new ScenarioInput("healthCheck", MESSAGES.demoHealthCheck, {
        type: "confirm",
      }),
      output: getHealthCheckResult,
    },
  },
);

```

```

const statusSteps = [
  getRecommendation,
  getRecommendationResult,
  getHealthCheck,
  getHealthCheckResult,
];

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const demoSteps = [
  new ScenarioOutput("header", MESSAGES.demoHeader, { header: true }),
  new ScenarioOutput("sanityCheck", MESSAGES.demoSanityCheck),
  ...statusSteps,
  new ScenarioInput(
    "brokenDependencyConfirmation",
    MESSAGES.demoBrokenDependencyConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("brokenDependency", async (state) => {
    if (!state.brokenDependencyConfirmation) {
      process.exit();
    } else {
      const client = new SSMClient({});
      state.badTableName = `fake-table-${Date.now()}`;
      await client.send(
        new PutParameterCommand({
          Name: NAMES.ssmTableNameKey,
          Value: state.badTableName,
          Overwrite: true,
          Type: "String",
        }),
      );
    }
  }),
  new ScenarioOutput("testBrokenDependency", (state) =>
    MESSAGES.demoTestBrokenDependency.replace(
      "${TABLE_NAME}",
      state.badTableName,
    ),
  ),
  ...statusSteps,
  new ScenarioInput(
    "staticResponseConfirmation",
  ),
];

```

```

    MESSAGES.demoStaticResponseConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("staticResponse", async (state) => {
    if (!state.staticResponseConfirmation) {
      process.exit();
    } else {
      const client = new SSMClient({});
      await client.send(
        new PutParameterCommand({
          Name: NAMES.ssmFailureResponseKey,
          Value: "static",
          Overwrite: true,
          Type: "String",
        }),
      );
    }
  }),
  new ScenarioOutput("testStaticResponse", MESSAGES.demoTestStaticResponse),
  ...statusSteps,
  new ScenarioInput(
    "badCredentialsConfirmation",
    MESSAGES.demoBadCredentialsConfirmation,
    { type: "confirm" },
  ),
  new ScenarioAction("badCredentialsExit", (state) => {
    if (!state.badCredentialsConfirmation) {
      process.exit();
    }
  }),
  new ScenarioAction("fixDynamoDBName", async () => {
    const client = new SSMClient({});
    await client.send(
      new PutParameterCommand({
        Name: NAMES.ssmTableNameKey,
        Value: NAMES.tableName,
        Overwrite: true,
        Type: "String",
      }),
    );
  }),
  new ScenarioAction(
    "badCredentials",
    /**

```

```

    * @param {{ targetInstance: import('@aws-sdk/client-auto-scaling').Instance }}
state
    */
    async (state) => {
        await createSsmOnlyInstanceProfile();
        const autoScalingClient = new AutoScalingClient({});
        const { AutoScalingGroups } = await autoScalingClient.send(
            new DescribeAutoScalingGroupsCommand({
                AutoScalingGroupNames: [NAMES.autoScalingGroupName],
            }),
        );
        state.targetInstance = AutoScalingGroups[0].Instances[0];
        const ec2Client = new EC2Client({});
        const { IamInstanceProfileAssociations } = await ec2Client.send(
            new DescribeIamInstanceProfileAssociationsCommand({
                Filters: [
                    { Name: "instance-id", Values: [state.targetInstance.InstanceId] },
                ],
            }),
        );
        state.instanceProfileAssociationId =
            IamInstanceProfileAssociations[0].AssociationId;
        await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
            ec2Client.send(
                new ReplaceIamInstanceProfileAssociationCommand({
                    AssociationId: state.instanceProfileAssociationId,
                    IamInstanceProfile: { Name: NAMES.ssmOnlyInstanceProfileName },
                }),
            ),
        );

        await ec2Client.send(
            new RebootInstancesCommand({
                InstanceIds: [state.targetInstance.InstanceId],
            }),
        );

        const ssmClient = new SSMClient({});
        await retry({ intervalInMs: 20000, maxRetries: 15 }, async () => {
            const { InstanceInformationList } = await ssmClient.send(
                new DescribeInstanceInformationCommand({}),
            );

            const instance = InstanceInformationList.find(

```

```

        (info) => info.InstanceId === state.targetInstance.InstanceId,
    );

    if (!instance) {
        throw new Error("Instance not found.");
    }
});

await ssmClient.send(
    new SendCommandCommand({
        InstanceIds: [state.targetInstance.InstanceId],
        DocumentName: "AWS-RunShellScript",
        Parameters: { commands: ["cd / && sudo python3 server.py 80"] },
    }),
);
},
),
new ScenarioOutput(
    "testBadCredentials",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-ssm').InstanceInformation}}
state
    */
    (state) =>
        MESSAGES.demoTestBadCredentials.replace(
            "${INSTANCE_ID}",
            state.targetInstance.InstanceId,
        ),
),
loadBalancerLoop,
new ScenarioInput(
    "deepHealthCheckConfirmation",
    MESSAGES.demoDeepHealthCheckConfirmation,
    { type: "confirm" },
),
new ScenarioAction("deepHealthCheckExit", (state) => {
    if (!state.deepHealthCheckConfirmation) {
        process.exit();
    }
}),
new ScenarioAction("deepHealthCheck", async () => {
    const client = new SSMClient({});
    await client.send(
        new PutParameterCommand({

```

```

        Name: NAMES.ssmHealthCheckKey,
        Value: "deep",
        Overwrite: true,
        Type: "String",
    })),
    );
  })),
  new ScenarioOutput("testDeepHealthCheck", MESSAGES.demoTestDeepHealthCheck),
  healthCheckLoop,
  loadBalancerLoop,
  new ScenarioInput(
    "killInstanceConfirmation",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-
    ssm').InstanceInformation }} state
     */
    (state) =>
      MESSAGES.demoKillInstanceConfirmation.replace(
        "${INSTANCE_ID}",
        state.targetInstance.InstanceId,
      ),
    { type: "confirm" },
  ),
  new ScenarioAction("killInstanceExit", (state) => {
    if (!state.killInstanceConfirmation) {
      process.exit();
    }
  })),
  new ScenarioAction(
    "killInstance",
    /**
     * @param {{ targetInstance: import('@aws-sdk/client-
    ssm').InstanceInformation }} state
     */
    async (state) => {
      const client = new AutoScalingClient({});
      await client.send(
        new TerminateInstanceInAutoScalingGroupCommand({
          InstanceId: state.targetInstance.InstanceId,
          ShouldDecrementDesiredCapacity: false,
        }),
      );
    },
  ),
),

```

```
new ScenarioOutput("testKillInstance", MESSAGES.demoTestKillInstance),
healthCheckLoop,
loadBalancerLoop,
new ScenarioInput("failOpenConfirmation", MESSAGES.demoFailOpenConfirmation, {
  type: "confirm",
}),
new ScenarioAction("failOpenExit", (state) => {
  if (!state.failOpenConfirmation) {
    process.exit();
  }
}),
new ScenarioAction("failOpen", () => {
  const client = new SSMClient({});
  return client.send(
    new PutParameterCommand({
      Name: NAMES.ssmTableNameKey,
      Value: `fake-table-${Date.now()}`,
      Overwrite: true,
      Type: "String",
    })
  );
}),
new ScenarioOutput("testFailOpen", MESSAGES.demoFailOpenTest),
healthCheckLoop,
loadBalancerLoop,
new ScenarioInput(
  "resetTableConfirmation",
  MESSAGES.demoResetTableConfirmation,
  { type: "confirm" },
),
new ScenarioAction("resetTableExit", (state) => {
  if (!state.resetTableConfirmation) {
    process.exit();
  }
}),
new ScenarioAction("resetTable", async () => {
  const client = new SSMClient({});
  await client.send(
    new PutParameterCommand({
      Name: NAMES.ssmTableNameKey,
      Value: NAMES.tableName,
      Overwrite: true,
      Type: "String",
    })
  );
}),
```

```
    );
  }},
  new ScenarioOutput("testResetTable", MESSAGES.demoTestResetTable),
  healthCheckLoop,
  loadBalancerLoop,
];

async function createSsmOnlyInstanceProfile() {
  const iamClient = new IAMClient({});
  const { Policy } = await iamClient.send(
    new CreatePolicyCommand({
      PolicyName: NAMES.ssmOnlyPolicyName,
      PolicyDocument: readFileSync(
        join(RESOURCES_PATH, "ssm_only_policy.json"),
      ),
    }),
  );
  await iamClient.send(
    new CreateRoleCommand({
      RoleName: NAMES.ssmOnlyRoleName,
      AssumeRolePolicyDocument: JSON.stringify({
        Version: "2012-10-17",
        Statement: [
          {
            Effect: "Allow",
            Principal: { Service: "ec2.amazonaws.com" },
            Action: "sts:AssumeRole",
          },
        ],
      }),
    }),
  );
  await iamClient.send(
    new AttachRolePolicyCommand({
      RoleName: NAMES.ssmOnlyRoleName,
      PolicyArn: Policy.Arn,
    }),
  );
  await iamClient.send(
    new AttachRolePolicyCommand({
      RoleName: NAMES.ssmOnlyRoleName,
      PolicyArn: "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore",
    }),
  );
}
```



```

const { InstanceProfile } = await iamClient.send(
  new CreateInstanceProfileCommand({
    InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
  }),
);
await waitUntilInstanceProfileExists(
  { client: iamClient },
  { InstanceProfileName: NAMES.ssmOnlyInstanceProfileName },
);
await iamClient.send(
  new AddRoleToInstanceProfileCommand({
    InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
    RoleName: NAMES.ssmOnlyRoleName,
  }),
);

return InstanceProfile;
}

```

Erstellen Sie Schritte, um alle Ressourcen zu vernichten.

```

import { unlinkSync } from "node:fs";

import { DynamoDBClient, DeleteTableCommand } from "@aws-sdk/client-dynamodb";
import {
  EC2Client,
  DeleteKeyPairCommand,
  DeleteLaunchTemplateCommand,
  RevokeSecurityGroupIngressCommand,
} from "@aws-sdk/client-ec2";
import {
  IAMClient,
  DeleteInstanceProfileCommand,
  RemoveRoleFromInstanceProfileCommand,
  DeletePolicyCommand,
  DeleteRoleCommand,
  DetachRolePolicyCommand,
  paginateListPolicies,
} from "@aws-sdk/client-iam";
import {
  AutoScalingClient,
  DeleteAutoScalingGroupCommand,

```

```

    TerminateInstanceInAutoScalingGroupCommand,
    UpdateAutoScalingGroupCommand,
    paginateDescribeAutoScalingGroups,
} from "@aws-sdk/client-auto-scaling";
import {
    DeleteLoadBalancerCommand,
    DeleteTargetGroupCommand,
    DescribeTargetGroupsCommand,
    ElasticLoadBalancingV2Client,
} from "@aws-sdk/client-elastic-load-balancing-v2";

import {
    ScenarioOutput,
    ScenarioInput,
    ScenarioAction,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { loadState } from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

import { MESSAGES, NAMES } from "../constants.js";
import { findLoadBalancer } from "../shared.js";

/**
 * @type {import('@aws-doc-sdk-examples/lib/scenario.js').Step[]}
 */
export const destroySteps = [
    loadState,
    new ScenarioInput("destroy", MESSAGES.destroy, { type: "confirm" }),
    new ScenarioAction(
        "abort",
        (state) => state.destroy === false && process.exit(),
    ),
    new ScenarioAction("deleteTable", async (c) => {
        try {
            const client = new DynamoDBClient({});
            await client.send(new DeleteTableCommand({ TableName: NAMES.tableName }));
        } catch (e) {
            c.deleteTableError = e;
        }
    }),
    new ScenarioOutput("deleteTableResult", (state) => {
        if (state.deleteTableError) {
            console.error(state.deleteTableError);
            return MESSAGES.deleteTableError.replace(

```

```

        "${TABLE_NAME}",
        NAMES.tableName,
    );
}
return MESSAGES.deletedTable.replace("${TABLE_NAME}", NAMES.tableName);
}),
new ScenarioAction("deleteKeyPair", async (state) => {
    try {
        const client = new EC2Client({});
        await client.send(
            new DeleteKeyPairCommand({ KeyName: NAMES.keyPairName }),
        );
        unlinkSync(`${NAMES.keyPairName}.pem`);
    } catch (e) {
        state.deleteKeyPairError = e;
    }
}),
new ScenarioOutput("deleteKeyPairResult", (state) => {
    if (state.deleteKeyPairError) {
        console.error(state.deleteKeyPairError);
        return MESSAGES.deleteKeyPairError.replace(
            "${KEY_PAIR_NAME}",
            NAMES.keyPairName,
        );
    }
    return MESSAGES.deletedKeyPair.replace(
        "${KEY_PAIR_NAME}",
        NAMES.keyPairName,
    );
}),
new ScenarioAction("detachPolicyFromRole", async (state) => {
    try {
        const client = new IAMClient({});
        const policy = await findPolicy(NAMES.instancePolicyName);

        if (!policy) {
            state.detachPolicyFromRoleError = new Error(
                `Policy ${NAMES.instancePolicyName} not found.`,
            );
        } else {
            await client.send(
                new DetachRolePolicyCommand({
                    RoleName: NAMES.instanceRoleName,
                    PolicyArn: policy.Arn,
                })
            );
        }
    }
});

```

```

        })),
    );
}
} catch (e) {
    state.detachPolicyFromRoleError = e;
}
})),
new ScenarioOutput("detachedPolicyFromRole", (state) => {
    if (state.detachPolicyFromRoleError) {
        console.error(state.detachPolicyFromRoleError);
        return MESSAGES.detachPolicyFromRoleError
            .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
            .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
    }
    return MESSAGES.detachedPolicyFromRole
        .replace("${INSTANCE_POLICY_NAME}", NAMES.instancePolicyName)
        .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
})),
new ScenarioAction("deleteInstancePolicy", async (state) => {
    const client = new IAMClient({});
    const policy = await findPolicy(NAMES.instancePolicyName);

    if (!policy) {
        state.deletePolicyError = new Error(
            `Policy ${NAMES.instancePolicyName} not found.`
        );
    } else {
        return client.send(
            new DeletePolicyCommand({
                PolicyArn: policy.Arn,
            })
        );
    }
})),
new ScenarioOutput("deletePolicyResult", (state) => {
    if (state.deletePolicyError) {
        console.error(state.deletePolicyError);
        return MESSAGES.deletePolicyError.replace(
            "${INSTANCE_POLICY_NAME}",
            NAMES.instancePolicyName,
        );
    }
    return MESSAGES.deletedPolicy.replace(
        "${INSTANCE_POLICY_NAME}",

```

```
    NAMES.instancePolicyName,
  );
}),
new ScenarioAction("removeRoleFromInstanceProfile", async (state) => {
  try {
    const client = new IAMClient({});
    await client.send(
      new RemoveRoleFromInstanceProfileCommand({
        RoleName: NAMES.instanceRoleName,
        InstanceProfileName: NAMES.instanceProfileName,
      }),
    );
  } catch (e) {
    state.removeRoleFromInstanceProfileError = e;
  }
}),
new ScenarioOutput("removeRoleFromInstanceProfileResult", (state) => {
  if (state.removeRoleFromInstanceProfile) {
    console.error(state.removeRoleFromInstanceProfileError);
    return MESSAGES.removeRoleFromInstanceProfileError
      .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
      .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
  }
  return MESSAGES.removedRoleFromInstanceProfile
    .replace("${INSTANCE_PROFILE_NAME}", NAMES.instanceProfileName)
    .replace("${INSTANCE_ROLE_NAME}", NAMES.instanceRoleName);
}),
new ScenarioAction("deleteInstanceRole", async (state) => {
  try {
    const client = new IAMClient({});
    await client.send(
      new DeleteRoleCommand({
        RoleName: NAMES.instanceRoleName,
      }),
    );
  } catch (e) {
    state.deleteInstanceRoleError = e;
  }
}),
new ScenarioOutput("deleteInstanceRoleResult", (state) => {
  if (state.deleteInstanceRoleError) {
    console.error(state.deleteInstanceRoleError);
    return MESSAGES.deleteInstanceRoleError.replace(
      "${INSTANCE_ROLE_NAME}",
```

```
        NAMES.instanceRoleName,
    );
}
return MESSAGES.deletedInstanceRole.replace(
    "${INSTANCE_ROLE_NAME}",
    NAMES.instanceRoleName,
);
}),
new ScenarioAction("deleteInstanceProfile", async (state) => {
    try {
        const client = new IAMClient({});
        await client.send(
            new DeleteInstanceProfileCommand({
                InstanceProfileName: NAMES.instanceProfileName,
            }),
        );
    } catch (e) {
        state.deleteInstanceProfileError = e;
    }
}),
new ScenarioOutput("deleteInstanceProfileResult", (state) => {
    if (state.deleteInstanceProfileError) {
        console.error(state.deleteInstanceProfileError);
        return MESSAGES.deleteInstanceProfileError.replace(
            "${INSTANCE_PROFILE_NAME}",
            NAMES.instanceProfileName,
        );
    }
    return MESSAGES.deletedInstanceProfile.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.instanceProfileName,
    );
}),
new ScenarioAction("deleteAutoScalingGroup", async (state) => {
    try {
        await terminateGroupInstances(NAMES.autoScalingGroupName);
        await retry({ intervalInMs: 60000, maxRetries: 60 }, async () => {
            await deleteAutoScalingGroup(NAMES.autoScalingGroupName);
        });
    } catch (e) {
        state.deleteAutoScalingGroupError = e;
    }
}),
new ScenarioOutput("deleteAutoScalingGroupResult", (state) => {
```

```
if (state.deleteAutoScalingGroupError) {
  console.error(state.deleteAutoScalingGroupError);
  return MESSAGES.deleteAutoScalingGroupError.replace(
    "${AUTO_SCALING_GROUP_NAME}",
    NAMES.autoScalingGroupName,
  );
}
return MESSAGES.deletedAutoScalingGroup.replace(
  "${AUTO_SCALING_GROUP_NAME}",
  NAMES.autoScalingGroupName,
);
}),
new ScenarioAction("deleteLaunchTemplate", async (state) => {
  const client = new EC2Client({});
  try {
    await client.send(
      new DeleteLaunchTemplateCommand({
        LaunchTemplateName: NAMES.launchTemplateName,
      }),
    );
  } catch (e) {
    state.deleteLaunchTemplateError = e;
  }
}),
new ScenarioOutput("deleteLaunchTemplateResult", (state) => {
  if (state.deleteLaunchTemplateError) {
    console.error(state.deleteLaunchTemplateError);
    return MESSAGES.deleteLaunchTemplateError.replace(
      "${LAUNCH_TEMPLATE_NAME}",
      NAMES.launchTemplateName,
    );
  }
  return MESSAGES.deletedLaunchTemplate.replace(
    "${LAUNCH_TEMPLATE_NAME}",
    NAMES.launchTemplateName,
  );
}),
new ScenarioAction("deleteLoadBalancer", async (state) => {
  try {
    const client = new ElasticLoadBalancingV2Client({});
    const loadBalancer = await findLoadBalancer(NAMES.loadBalancerName);
    await client.send(
      new DeleteLoadBalancerCommand({
        LoadBalancerArn: loadBalancer.LoadBalancerArn,
```

```

    }},
  );
  await retry({ intervalInMs: 1000, maxRetries: 60 }, async () => {
    const lb = await findLoadBalancer(NAMES.loadBalancerName);
    if (lb) {
      throw new Error("Load balancer still exists.");
    }
  });
} catch (e) {
  state.deleteLoadBalancerError = e;
}
}),
new ScenarioOutput("deleteLoadBalancerResult", (state) => {
  if (state.deleteLoadBalancerError) {
    console.error(state.deleteLoadBalancerError);
    return MESSAGES.deleteLoadBalancerError.replace(
      "${LB_NAME}",
      NAMES.loadBalancerName,
    );
  }
  return MESSAGES.deletedLoadBalancer.replace(
    "${LB_NAME}",
    NAMES.loadBalancerName,
  );
}),
new ScenarioAction("deleteLoadBalancerTargetGroup", async (state) => {
  const client = new ElasticLoadBalancingV2Client({});
  try {
    const { TargetGroups } = await client.send(
      new DescribeTargetGroupsCommand({
        Names: [NAMES.loadBalancerTargetGroupName],
      }),
    );
  };

  await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
    client.send(
      new DeleteTargetGroupCommand({
        TargetGroupArn: TargetGroups[0].TargetGroupArn,
      }),
    ),
  );
} catch (e) {
  state.deleteLoadBalancerTargetGroupError = e;
}

```



```

    }},
    new ScenarioOutput("deleteLoadBalancerTargetGroupResult", (state) => {
      if (state.deleteLoadBalancerTargetGroupError) {
        console.error(state.deleteLoadBalancerTargetGroupError);
        return MESSAGES.deleteLoadBalancerTargetGroupError.replace(
          "${TARGET_GROUP_NAME}",
          NAMES.loadBalancerTargetGroupName,
        );
      }
      return MESSAGES.deletedLoadBalancerTargetGroup.replace(
        "${TARGET_GROUP_NAME}",
        NAMES.loadBalancerTargetGroupName,
      );
    }),
    new ScenarioAction("detachSsmOnlyRoleFromProfile", async (state) => {
      try {
        const client = new IAMClient({});
        await client.send(
          new RemoveRoleFromInstanceProfileCommand({
            InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
            RoleName: NAMES.ssmOnlyRoleName,
          }),
        );
      } catch (e) {
        state.detachSsmOnlyRoleFromProfileError = e;
      }
    }),
    new ScenarioOutput("detachSsmOnlyRoleFromProfileResult", (state) => {
      if (state.detachSsmOnlyRoleFromProfileError) {
        console.error(state.detachSsmOnlyRoleFromProfileError);
        return MESSAGES.detachSsmOnlyRoleFromProfileError
          .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
          .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
      }
      return MESSAGES.detachedSsmOnlyRoleFromProfile
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${PROFILE_NAME}", NAMES.ssmOnlyInstanceProfileName);
    }),
    new ScenarioAction("detachSsmOnlyCustomRolePolicy", async (state) => {
      try {
        const iamClient = new IAMClient({});
        const ssmOnlyPolicy = await findPolicy(NAMES.ssmOnlyPolicyName);
        await iamClient.send(
          new DetachRolePolicyCommand({

```

```

        RoleName: NAMES.ssmOnlyRoleName,
        PolicyArn: ssmOnlyPolicy.Arn,
    })),
    );
} catch (e) {
    state.detachSsmOnlyCustomRolePolicyError = e;
}
}),
new ScenarioOutput("detachSsmOnlyCustomRolePolicyResult", (state) => {
    if (state.detachSsmOnlyCustomRolePolicyError) {
        console.error(state.detachSsmOnlyCustomRolePolicyError);
        return MESSAGES.detachSsmOnlyCustomRolePolicyError
            .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
            .replace("${POLICY_NAME}", NAMES.ssmOnlyPolicyName);
    }
    return MESSAGES.detachedSsmOnlyCustomRolePolicy
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", NAMES.ssmOnlyPolicyName);
}),
new ScenarioAction("detachSsmOnlyAWSRolePolicy", async (state) => {
    try {
        const iamClient = new IAMClient({});
        await iamClient.send(
            new DetachRolePolicyCommand({
                RoleName: NAMES.ssmOnlyRoleName,
                PolicyArn: "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore",
            }),
        );
    } catch (e) {
        state.detachSsmOnlyAWSRolePolicyError = e;
    }
}),
new ScenarioOutput("detachSsmOnlyAWSRolePolicyResult", (state) => {
    if (state.detachSsmOnlyAWSRolePolicyError) {
        console.error(state.detachSsmOnlyAWSRolePolicyError);
        return MESSAGES.detachSsmOnlyAWSRolePolicyError
            .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
            .replace("${POLICY_NAME}", "AmazonSSMManagedInstanceCore");
    }
    return MESSAGES.detachedSsmOnlyAWSRolePolicy
        .replace("${ROLE_NAME}", NAMES.ssmOnlyRoleName)
        .replace("${POLICY_NAME}", "AmazonSSMManagedInstanceCore");
}),
new ScenarioAction("deleteSsmOnlyInstanceProfile", async (state) => {

```

```
    try {
      const iamClient = new IAMClient({});
      await iamClient.send(
        new DeleteInstanceProfileCommand({
          InstanceProfileName: NAMES.ssmOnlyInstanceProfileName,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyInstanceProfileError = e;
    }
  })),
  new ScenarioOutput("deleteSsmOnlyInstanceProfileResult", (state) => {
    if (state.deleteSsmOnlyInstanceProfileError) {
      console.error(state.deleteSsmOnlyInstanceProfileError);
      return MESSAGES.deleteSsmOnlyInstanceProfileError.replace(
        "${INSTANCE_PROFILE_NAME}",
        NAMES.ssmOnlyInstanceProfileName,
      );
    }
    return MESSAGES.deletedSsmOnlyInstanceProfile.replace(
      "${INSTANCE_PROFILE_NAME}",
      NAMES.ssmOnlyInstanceProfileName,
    );
  })),
  new ScenarioAction("deleteSsmOnlyPolicy", async (state) => {
    try {
      const iamClient = new IAMClient({});
      const ssmOnlyPolicy = await findPolicy(NAMES.ssmOnlyPolicyName);
      await iamClient.send(
        new DeletePolicyCommand({
          PolicyArn: ssmOnlyPolicy.Arn,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyPolicyError = e;
    }
  })),
  new ScenarioOutput("deleteSsmOnlyPolicyResult", (state) => {
    if (state.deleteSsmOnlyPolicyError) {
      console.error(state.deleteSsmOnlyPolicyError);
      return MESSAGES.deleteSsmOnlyPolicyError.replace(
        "${POLICY_NAME}",
        NAMES.ssmOnlyPolicyName,
      );
    }
  })
```

```

    }
    return MESSAGES.deletedSsmOnlyPolicy.replace(
      "${POLICY_NAME}",
      NAMES.ssmOnlyPolicyName,
    );
  }},
  new ScenarioAction("deleteSsmOnlyRole", async (state) => {
    try {
      const iamClient = new IAMClient({});
      await iamClient.send(
        new DeleteRoleCommand({
          RoleName: NAMES.ssmOnlyRoleName,
        }),
      );
    } catch (e) {
      state.deleteSsmOnlyRoleError = e;
    }
  }},
  new ScenarioOutput("deleteSsmOnlyRoleResult", (state) => {
    if (state.deleteSsmOnlyRoleError) {
      console.error(state.deleteSsmOnlyRoleError);
      return MESSAGES.deleteSsmOnlyRoleError.replace(
        "${ROLE_NAME}",
        NAMES.ssmOnlyRoleName,
      );
    }
    return MESSAGES.deletedSsmOnlyRole.replace(
      "${ROLE_NAME}",
      NAMES.ssmOnlyRoleName,
    );
  }},
  new ScenarioAction(
    "revokeSecurityGroupIngress",
    async (
      /** @type {{ myIp: string, defaultSecurityGroup: { GroupId: string } }} */
      state,
    ) => {
      const ec2Client = new EC2Client({});

      try {
        await ec2Client.send(
          new RevokeSecurityGroupIngressCommand({
            GroupId: state.defaultSecurityGroup.GroupId,
            CidrIp: `${state.myIp}/32`,

```

```

        FromPort: 80,
        ToPort: 80,
        IpProtocol: "tcp",
      )),
    );
  } catch (e) {
    state.revokeSecurityGroupIngressError = e;
  }
},
),
new ScenarioOutput("revokeSecurityGroupIngressResult", (state) => {
  if (state.revokeSecurityGroupIngressError) {
    console.error(state.revokeSecurityGroupIngressError);
    return MESSAGES.revokeSecurityGroupIngressError.replace(
      "${IP}",
      state.myIp,
    );
  }
  return MESSAGES.revokedSecurityGroupIngress.replace("${IP}", state.myIp);
}),
];

/**
 * @param {string} policyName
 */
async function findPolicy(policyName) {
  const client = new IAMClient({});
  const paginatedPolicies = paginateListPolicies({ client }, {});
  for await (const page of paginatedPolicies) {
    const policy = page.Policies.find((p) => p.PolicyName === policyName);
    if (policy) {
      return policy;
    }
  }
}

/**
 * @param {string} groupName
 */
async function deleteAutoScalingGroup(groupName) {
  const client = new AutoScalingClient({});
  try {
    await client.send(
      new DeleteAutoScalingGroupCommand({

```

```

        AutoScalingGroupName: groupName,
    })),
    );
} catch (err) {
    if (!(err instanceof Error)) {
        throw err;
    }
    console.log(err.name);
    throw err;
}
}

/**
 * @param {string} groupName
 */
async function terminateGroupInstances(groupName) {
    const autoScalingClient = new AutoScalingClient({});
    const group = await findAutoScalingGroup(groupName);
    await autoScalingClient.send(
        new UpdateAutoScalingGroupCommand({
            AutoScalingGroupName: group.AutoScalingGroupName,
            MinSize: 0,
        })),
    );
    for (const i of group.Instances) {
        await retry({ intervalInMs: 1000, maxRetries: 30 }, () =>
            autoScalingClient.send(
                new TerminateInstanceInAutoScalingGroupCommand({
                    InstanceId: i.InstanceId,
                    ShouldDecrementDesiredCapacity: true,
                })),
        );
    }
}

async function findAutoScalingGroup(groupName) {
    const client = new AutoScalingClient({});
    const paginatedGroups = paginateDescribeAutoScalingGroups({ client }, {});
    for await (const page of paginatedGroups) {
        const group = page.AutoScalingGroups.find(
            (g) => g.AutoScalingGroupName === groupName,
        );
        if (group) {

```

```
    return group;
  }
}
throw new Error(`Auto scaling group ${groupName} not found.`);
}
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für JavaScript - API-Referenz.

- [AttachLoadBalancerTargetGroups](#)
- [CreateAutoScalingGroup](#)
- [CreateInstanceProfile](#)
- [CreateLaunchTemplate](#)
- [CreateListener](#)
- [CreateLoadBalancer](#)
- [CreateTargetGroup](#)
- [DeleteAutoScalingGroup](#)
- [DeleteInstanceProfile](#)
- [DeleteLaunchTemplate](#)
- [DeleteLoadBalancer](#)
- [DeleteTargetGroup](#)
- [DescribeAutoScalingGroups](#)
- [DescribeAvailabilityZones](#)
- [DescribeIamInstanceProfileAssociations](#)
- [DescribeInstances](#)
- [DescribeLoadBalancers](#)
- [DescribeSubnets](#)
- [DescribeTargetGroups](#)
- [DescribeTargetHealth](#)
- [DescribeVpcs](#)
- [RebootInstances](#)

- [UpdateAutoScalingGroup](#)

AWS IoT SiteWise Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie AWS SDK für JavaScript (v3) mit verwenden AWS IoT SiteWise.

Bei Grundlagen handelt es sich um Codebeispiele, die Ihnen zeigen, wie Sie die wesentlichen Vorgänge innerhalb eines Services ausführen.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kodex finden.

Themen

- [Erste Schritte](#)
- [Grundlagen](#)
- [Aktionen](#)

Erste Schritte

Hallo AWS IoT SiteWise

Das folgende Codebeispiel veranschaulicht, wie Sie mit der Verwendung von AWS IoT SiteWise beginnen.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import {
```



```
    paginateListAssetModels,
    IoTSiteWiseClient,
  } from "@aws-sdk/client-iotsitewise";

// Call ListDocuments and display the result.
export const main = async () => {
  const client = new IoTSiteWiseClient();
  const listAssetModelsPaginated = [];
  console.log(
    "Hello, AWS Systems Manager! Let's list some of your documents:\n",
  );
  try {
    // The paginate function is a wrapper around the base command.
    const paginator = paginateListAssetModels({ client }, { maxResults: 5 });
    for await (const page of paginator) {
      listAssetModelsPaginated.push(...page.assetModelSummaries);
    }
  } catch (caught) {
    console.error(`There was a problem saying hello: ${caught.message}`);
    throw caught;
  }
  for (const { name, creationDate } of listAssetModelsPaginated) {
    console.log(`${name} - ${creationDate}`);
  }
};

// Call function if run directly.
import { fileURLToPath } from "node:url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  main();
}
```

- Einzelheiten zur API finden Sie [ListAssetModels](#) in der AWS SDK für JavaScript API-Referenz.

Grundlagen

Kennenlernen der Grundlagen

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Erstellen Sie ein AWS IoT SiteWise Asset-Modell.

- Erstellen Sie ein AWS IoT SiteWise Asset.
- Rufen Sie die Eigenschafts-ID-Werte ab.
- Daten an ein AWS IoT SiteWise Asset senden.
- Ruft den Wert der Eigenschaft AWS IoT SiteWise Asset ab.
- Erstellen Sie ein AWS IoT SiteWise Portal.
- Erstellen Sie ein AWS IoT SiteWise Gateway.
- Beschreiben Sie das AWS IoT SiteWise Gateway.
- Löschen Sie die AWS IoT SiteWise Assets.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import {
  Scenario,
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
  //} from "@aws-doc-sdk-examples/lib/scenario/index.js";
} from "../../libs/scenario/index.js";
import {
  IoTSiteWiseClient,
  CreateAssetModelCommand,
  CreateAssetCommand,
  ListAssetModelPropertiesCommand,
  BatchPutAssetPropertyValueCommand,
  GetAssetPropertyValueCommand,
  CreatePortalCommand,
  DescribePortalCommand,
  CreateGatewayCommand,
  DescribeGatewayCommand,
  DeletePortalCommand,
  DeleteGatewayCommand,
  DeleteAssetCommand,
  DeleteAssetModelCommand,
```

```

    DescribeAssetModelCommand,
  } from "@aws-sdk/client-iotsitewise";
import {
  CloudFormationClient,
  CreateStackCommand,
  DeleteStackCommand,
  DescribeStacksCommand,
  waitUntilStackExists,
  waitUntilStackCreateComplete,
  waitUntilStackDeleteComplete,
} from "@aws-sdk/client-cloudformation";
import { wait } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";
import { parseArgs } from "node:util";
import { readFileSync } from "node:fs";
import { fileURLToPath } from "node:url";
import { dirname } from "node:path";

const __filename = fileURLToPath(import.meta.url);
const __dirname = dirname(__filename);
const stackName = "SiteWiseBasicsStack";

/**
 * @typedef {{
 *   iotSiteWiseClient: import('@aws-sdk/client-iotsitewise').IotSiteWiseClient,
 *   cloudFormationClient: import('@aws-sdk/client-cloudformation').CloudFormationClient,
 *   stackName,
 *   stack,
 *   askToDeleteResources: true,
 *   asset: {assetName: "MyAsset1"},
 *   assetModel: {assetModelName: "MyAssetModel1"},
 *   portal: {portalName: "MyPortal1"},
 *   gateway: {gatewayName: "MyGateway1"},
 *   propertyIds: [],
 *   contactEmail: "user@mydomain.com",
 *   thing: "MyThing1",
 *   sampleData: { temperature: 23.5, humidity: 65.0}
 * }} State
 */

/**
 * Used repeatedly to have the user press enter.
 * @type {ScenarioInput}
 */

```

```
const pressEnter = new ScenarioInput("continue", "Press Enter to continue", {
  type: "confirm",
});

const greet = new ScenarioOutput(
  "greet",
  `AWS IoT SiteWise is a fully managed industrial software-as-a-service (SaaS)
  that makes it easy to collect, store, organize, and monitor data from industrial
  equipment and processes. It is designed to help industrial and manufacturing
  organizations collect data from their equipment and processes, and use that data to
  make informed decisions about their operations.
  One of the key features of AWS IoT SiteWise is its ability to connect to a wide
  range of industrial equipment and systems, including programmable logic controllers
  (PLCs), sensors, and other industrial devices. It can collect data from these
  devices and organize it into a unified data model, making it easier to analyze and
  gain insights from the data. AWS IoT SiteWise also provides tools for visualizing
  the data, setting up alarms and alerts, and generating reports.
  Another key feature of AWS IoT SiteWise is its ability to scale to handle large
  volumes of data. It can collect and store data from thousands of devices and
  process millions of data points per second, making it suitable for large-scale
  industrial operations. Additionally, AWS IoT SiteWise is designed to be secure
  and compliant, with features like role-based access controls, data encryption,
  and integration with other AWS services for additional security and compliance
  features.

  Let's get started...`,
  { header: true },
);

const displayBuildCloudFormationStack = new ScenarioOutput(
  "displayBuildCloudFormationStack",
  "This scenario uses AWS CloudFormation to create an IAM role that is required for
  this scenario. The stack will now be deployed.",
);

const sdkBuildCloudFormationStack = new ScenarioAction(
  "sdkBuildCloudFormationStack",
  async (** @type {State} */ state) => {
    try {
      const data = readFileSync(
        `${__dirname}/../../../../resources/cfn/iotsitewise_basics/SitewiseRoles-
        template.yml`,
        "utf8",
      );
    }
  });
```

```

    await state.cloudFormationClient.send(
      new CreateStackCommand({
        StackName: stackName,
        TemplateBody: data,
        Capabilities: ["CAPABILITY_IAM"],
      }),
    );
    await waitUntilStackExists(
      { client: state.cloudFormationClient },
      { StackName: stackName },
    );
    await waitUntilStackCreateComplete(
      { client: state.cloudFormationClient },
      { StackName: stackName },
    );
    const stack = await state.cloudFormationClient.send(
      new DescribeStacksCommand({
        StackName: stackName,
      }),
    );
    state.stack = stack.Stacks[0].Outputs[0];
    console.log(`The ARN of the IAM role is ${state.stack.OutputValue}`);
  } catch (caught) {
    console.error(caught.message);
    throw caught;
  }
},
);

const displayCreateAWSSiteWiseAssetModel = new ScenarioOutput(
  "displayCreateAWSSiteWiseAssetModel",
  `1. Create an AWS SiteWise Asset Model
An AWS IoT SiteWise Asset Model is a way to represent the physical assets, such
as equipment, processes, and systems, that exist in an industrial environment.
This model provides a structured and hierarchical representation of these assets,
allowing users to define the relationships and properties of each asset.

This scenario creates two asset model properties: temperature and humidity.`
);

const sdkCreateAWSSiteWiseAssetModel = new ScenarioAction(
  "sdkCreateAWSSiteWiseAssetModel",
  async (** @type {State} */ state) => {
    let assetModelResponse;

```

```

    try {
      assetModelResponse = await state.iotSiteWiseClient.send(
        new CreateAssetModelCommand({
          assetModelName: state.assetModel.assetModelName,
          assetModelProperties: [
            {
              name: "Temperature",
              dataType: "DOUBLE",
              type: {
                measurement: {},
              },
            },
            {
              name: "Humidity",
              dataType: "DOUBLE",
              type: {
                measurement: {},
              },
            },
          ],
        })
      );
      state.assetModel.assetModelId = assetModelResponse.assetModelId;
      console.log(
        `Asset Model successfully created. Asset Model ID:
        ${state.assetModel.assetModelId}`,
      );
    } catch (caught) {
      if (caught.name === "ResourceAlreadyExistsException") {
        console.log(
          `The Asset Model ${state.assetModel.assetModelName} already exists.`,
        );
        throw caught;
      }
      console.error(`${caught.message}`);
      throw caught;
    }
  },
);

const displayCreateAWSIoTSiteWiseAssetModel = new ScenarioOutput(
  "displayCreateAWSIoTSiteWiseAssetModel",
  `2. Create an AWS IoT SiteWise Asset

```

The IoT SiteWise model that we just created defines the structure and metadata for your physical assets. Now we create an asset from the asset model.

```
Let's wait 30 seconds for the asset to be ready.`,  
);  
  
const waitThirtySeconds = new ScenarioAction("waitThirtySeconds", async () => {  
  await wait(30); // wait 30 seconds  
  console.log("Time's up! Let's check the asset's status.");  
});
```

```
const sdkCreateAWSIoTSiteWiseAssetModel = new ScenarioAction(  
  "sdkCreateAWSIoTSiteWiseAssetModel",  
  async (/** @type {State} */ state) => {  
    try {  
      const assetResponse = await state.iotSiteWiseClient.send(  
        new CreateAssetCommand({  
          assetModelId: state.assetModel.assetModelId,  
          assetName: state.asset.assetName,  
        })),  
      );  
      state.asset.assetId = assetResponse.assetId;  
      console.log(`Asset created with ID: ${state.asset.assetId}`);  
    } catch (caught) {  
      if (caught.name === "ResourceNotFoundException") {  
        console.log(  
          `The Asset ${state.assetModel.assetModelName} was not found.`,  
        );  
        throw caught;  
      }  
      console.error(`${caught.message}`);  
      throw caught;  
    }  
  },  
);
```

```
const displayRetrievePropertyId = new ScenarioOutput(  
  "displayRetrievePropertyId",  
  `3. Retrieve the property ID values
```

To send data to an asset, we need to get the property ID values. In this scenario, we access the temperature and humidity property ID values.`,
);

```
const sdkRetrievePropertyId = new ScenarioAction(
  "sdkRetrievePropertyId",
  async (state) => {
    try {
      const retrieveResponse = await state.iotSiteWiseClient.send(
        new ListAssetModelPropertiesCommand({
          assetModelId: state.assetModel.assetModelId,
        }),
      );
      for (const retrieveResponseKey in
retrieveResponse.assetModelPropertySummaries) {
        if (
          retrieveResponse.assetModelPropertySummaries[retrieveResponseKey]
            .name === "Humidity"
        ) {
          state.propertyIds.Humidity =
            retrieveResponse.assetModelPropertySummaries[
              retrieveResponseKey
            ].id;
        }
        if (
          retrieveResponse.assetModelPropertySummaries[retrieveResponseKey]
            .name === "Temperature"
        ) {
          state.propertyIds.Temperature =
            retrieveResponse.assetModelPropertySummaries[
              retrieveResponseKey
            ].id;
        }
      }
      console.log(`The Humidity propertyId is ${state.propertyIds.Humidity}`);
      console.log(
        `The Temperature propertyId is ${state.propertyIds.Temperature}`,
      );
    } catch (caught) {
      if (caught.name === "IoTSiteWiseException") {
        console.log(
          `There was a problem retrieving the properties: ${caught.message}`,
        );
        throw caught;
      }
      console.error(`${caught.message}`);
      throw caught;
    }
  }
}
```



```

    },
  );

const displaySendDataToIoTSiteWiseAsset = new ScenarioOutput(
  "displaySendDataToIoTSiteWiseAsset",
  `4. Send data to an AWS IoT SiteWise Asset

```

By sending data to an IoT SiteWise Asset, you can aggregate data from multiple sources, normalize the data into a standard format, and store it in a centralized location. This makes it easier to analyze and gain insights from the data.

In this example, we generate sample temperature and humidity data and send it to the AWS IoT SiteWise asset.`,

```

  );

const sdkSendDataToIoTSiteWiseAsset = new ScenarioAction(
  "sdkSendDataToIoTSiteWiseAsset",
  async (state) => {
    try {
      const sendResponse = await state.iotSiteWiseClient.send(
        new BatchPutAssetPropertyValueCommand({
          entries: [
            {
              entryId: "entry-3",
              assetId: state.asset.assetId,
              propertyId: state.propertyIds.Humidity,
              propertyValues: [
                {
                  value: {
                    doubleValue: state.sampleData.humidity,
                  },
                  timestamp: {
                    timeInSeconds: Math.floor(Date.now() / 1000),
                  },
                },
              ],
            },
            {
              entryId: "entry-4",
              assetId: state.asset.assetId,
              propertyId: state.propertyIds.Temperature,
              propertyValues: [
                {
                  value: {

```

```

        doubleValue: state.sampleData.temperature,
      },
      timestamp: {
        timeInSeconds: Math.floor(Date.now() / 1000),
      },
    },
  ],
},
]),
}),
);
console.log("The data was sent successfully.");
} catch (caught) {
  if (caught.name === "ResourceNotFoundException") {
    console.log(`The Asset ${state.asset.assetName} was not found.`);
    throw caught;
  }
  console.error(`${caught.message}`);
  throw caught;
}
},
);

```

```

const displayRetrieveValueOfIoTSiteWiseAsset = new ScenarioOutput(
  "displayRetrieveValueOfIoTSiteWiseAsset",
  `5. Retrieve the value of the IoT SiteWise Asset property

```

IoT SiteWise is an AWS service that allows you to collect, process, and analyze industrial data from connected equipment and sensors. One of the key benefits of reading an IoT SiteWise property is the ability to gain valuable insights from your industrial data.`,

```

);

```

```

const sdkRetrieveValueOfIoTSiteWiseAsset = new ScenarioAction(
  "sdkRetrieveValueOfIoTSiteWiseAsset",
  async (** @type {State} */ state) => {
    try {
      const temperatureResponse = await state.iotSiteWiseClient.send(
        new GetAssetPropertyValueCommand({
          assetId: state.asset.assetId,
          propertyId: state.propertyIds.Temperature,
        }),
      );
      const humidityResponse = await state.iotSiteWiseClient.send(

```

```

        new GetAssetPropertyValueCommand({
            assetId: state.asset.assetId,
            propertyId: state.propertyIds.Humidity,
        }),
    );
    console.log(
        `The property value for Temperature is
    ${temperatureResponse.propertyValue.value.doubleValue}`,
    );
    console.log(
        `The property value for Humidity is
    ${humidityResponse.propertyValue.value.doubleValue}`,
    );
} catch (caught) {
    if (caught.name === "ResourceNotFoundException") {
        console.log(`The Asset ${state.asset.assetName} was not found.`);
        throw caught;
    }
    console.error(`${caught.message}`);
    throw caught;
}
},
);

```

```

const displayCreateIoTSiteWisePortal = new ScenarioOutput(
    "displayCreateIoTSiteWisePortal",
    `6. Create an IoT SiteWise Portal

```

An IoT SiteWise Portal allows you to aggregate data from multiple industrial sources, such as sensors, equipment, and control systems, into a centralized platform.`,

```

);

```

```

const sdkCreateIoTSiteWisePortal = new ScenarioAction(
    "sdkCreateIoTSiteWisePortal",
    async (/** @type {State} */ state) => {
        try {
            const createPortalResponse = await state.iotSiteWiseClient.send(
                new CreatePortalCommand({
                    portalName: state.portal.portalName,
                    portalContactEmail: state.contactEmail,
                    roleArn: state.stack.OutputValue,
                }),
            );
        }
    });

```

```

    state.portal = { ...state.portal, ...createPortalResponse };
    await wait(5); // Allow the portal to properly propagate.
    console.log(
      `Portal created successfully. Portal ID ${createPortalResponse.portalId}`,
    );
  } catch (caught) {
    if (caught.name === "IoTSiteWiseException") {
      console.log(
        `There was a problem creating the Portal: ${caught.message}.`,
      );
      throw caught;
    }
    console.error(`${caught.message}`);
    throw caught;
  }
},
);

const displayDescribePortal = new ScenarioOutput(
  "displayDescribePortal",
  `7. Describe the Portal

In this step, we get a description of the portal and display the portal URL.`
);

const sdkDescribePortal = new ScenarioAction(
  "sdkDescribePortal",
  async (** @type {State} */ state) => {
    try {
      const describePortalResponse = await state.iotSiteWiseClient.send(
        new DescribePortalCommand({
          portalId: state.portal.portalId,
        }),
      );
      console.log(`Portal URL: ${describePortalResponse.portalStartUrl}`);
    } catch (caught) {
      if (caught.name === "ResourceNotFoundException") {
        console.log(`The Portal ${state.portal.portalName} was not found.`);
        throw caught;
      }
      console.error(`${caught.message}`);
      throw caught;
    }
  },
),

```

```

);

const displayCreateIoTSiteWiseGateway = new ScenarioOutput(
  "displayCreateIoTSiteWiseGateway",
  `8. Create an IoT SiteWise Gateway

IoT SiteWise Gateway serves as the bridge between industrial equipment, sensors, and
the cloud-based IoT SiteWise service. It is responsible for securely collecting,
processing, and transmitting data from various industrial assets to the IoT
SiteWise platform, enabling real-time monitoring, analysis, and optimization of
industrial operations.`
);

const sdkCreateIoTSiteWiseGateway = new ScenarioAction(
  "sdkCreateIoTSiteWiseGateway",
  async (** @type {State} */ state) => {
    try {
      const createGatewayResponse = await state.iotSiteWiseClient.send(
        new CreateGatewayCommand({
          gatewayName: state.gateway.gatewayName,
          gatewayPlatform: {
            greengrassV2: {
              coreDeviceThingName: state.thing,
            },
          },
        })),
      );
      console.log(
        `Gateway creation completed successfully. ID is
${createGatewayResponse.gatewayId}`,
      );
      state.gateway.gatewayId = createGatewayResponse.gatewayId;
    } catch (caught) {
      if (caught.name === "IoTSiteWiseException") {
        console.log(
          `There was a problem creating the gateway: ${caught.message}.`,
        );
        throw caught;
      }
      console.error(`${caught.message}`);
      throw caught;
    }
  },
);

```

```

const displayDescribeIoTSiteWiseGateway = new ScenarioOutput(
  "displayDescribeIoTSiteWiseGateway",
  "9. Describe the IoT SiteWise Gateway",
);

const sdkDescribeIoTSiteWiseGateway = new ScenarioAction(
  "sdkDescribeIoTSiteWiseGateway",
  async (** @type {State} */ state) => {
    try {
      const describeGatewayResponse = await state.iotSiteWiseClient.send(
        new DescribeGatewayCommand({
          gatewayId: state.gateway.gatewayId,
        }),
      );
      console.log("Gateway creation completed successfully.");
      console.log(`Gateway Name: ${describeGatewayResponse.gatewayName}`);
      console.log(`Gateway ARN: ${describeGatewayResponse.gatewayArn}`);
      console.log(
        `Gateway Platform: ${Object.keys(describeGatewayResponse.gatewayPlatform)}`,
      );
      console.log(
        `Gateway Creation Date: ${describeGatewayResponse.creationDate}`,
      );
    } catch (caught) {
      if (caught.name === "ResourceNotFoundException") {
        console.log(`The Gateway ${state.gateway.gatewayId} was not found.`);
        throw caught;
      }
      console.error(`${caught.message}`);
      throw caught;
    }
  },
);

const askToDeleteResources = new ScenarioInput(
  "askToDeleteResources",
  `10. Delete the AWS IoT SiteWise Assets

Before you can delete the Asset Model, you must delete the assets.`,
  { type: "confirm" },
);

const displayConfirmDeleteResources = new ScenarioAction(

```

```
"displayConfirmDeleteResources",
async (** @type {State} */ state) => {
  if (state.askToDeleteResources) {
    return "You selected to delete the SiteWise assets.";
  }
  return "The resources will not be deleted. Please delete them manually to avoid charges.";
},
);

const sdkDeleteResources = new ScenarioAction(
  "sdkDeleteResources",
  async (** @type {State} */ state) => {
    await wait(10); // Give the portal status time to catch up.
    try {
      await state.iotSiteWiseClient.send(
        new DeletePortalCommand({
          portalId: state.portal.portalId,
        }),
      );
      console.log(
        `Portal ${state.portal.portalName} was deleted successfully.`,
      );
    } catch (caught) {
      if (caught.name === "ResourceNotFoundException") {
        console.log(`The Portal ${state.portal.portalName} was not found.`);
      } else {
        console.log(`When trying to delete the portal: ${caught.message}`);
      }
    }
  }

  try {
    await state.iotSiteWiseClient.send(
      new DeleteGatewayCommand({
        gatewayId: state.gateway.gatewayId,
      }),
    );
    console.log(
      `Gateway ${state.gateway.gatewayName} was deleted successfully.`,
    );
  } catch (caught) {
    if (caught.name === "ResourceNotFoundException") {
      console.log(`The Gateway ${state.gateway.gatewayId} was not found.`);
    } else {
```

```
        console.log(`When trying to delete the gateway: ${caught.message}`);
    }
}

try {
    await state.iotSiteWiseClient.send(
        new DeleteAssetCommand({
            assetId: state.asset.assetId,
        }),
    );
    await wait(5); // Allow the delete to finish.
    console.log(`Asset ${state.asset.assetName} was deleted successfully.`);
} catch (caught) {
    if (caught.name === "ResourceNotFoundException") {
        console.log(`The Asset ${state.asset.assetName} was not found.`);
    } else {
        console.log(`When deleting the asset: ${caught.message}`);
    }
}

await wait(30); // Allow asset deletion to finish.
try {
    await state.iotSiteWiseClient.send(
        new DeleteAssetModelCommand({
            assetModelId: state.assetModel.assetModelId,
        }),
    );
    console.log(
        `Asset Model ${state.assetModel.assetModelName} was deleted successfully.`,
    );
} catch (caught) {
    if (caught.name === "ResourceNotFoundException") {
        console.log(
            `The Asset Model ${state.assetModel.assetModelName} was not found.`,
        );
    } else {
        console.log(`When deleting the asset model: ${caught.message}`);
    }
}

try {
    await state.cloudFormationClient.send(
        new DeleteStackCommand({
            StackName: stackName,
```



```
    }},
  );
  await waitUntilStackDeleteComplete(
    { client: state.cloudFormationClient },
    { StackName: stackName },
  );
  console.log("The stack was deleted successfully.");
} catch (caught) {
  console.log(
    `${caught.message}. The stack was NOT deleted. Please clean up the resources manually.`
  );
}
},
{ skipWhen: (/** @type {} */ state) => !state.askToDeleteResources },
);

const goodbye = new ScenarioOutput(
  "goodbye",
  "This concludes the IoT Sitewise Basics scenario for the AWS Javascript SDK v3. Thank you!",
);

const myScenario = new Scenario(
  "IoTSiteWise Basics",
  [
    greet,
    pressEnter,
    displayBuildCloudFormationStack,
    sdkBuildCloudFormationStack,
    pressEnter,
    displayCreateAWSSiteWiseAssetModel,
    sdkCreateAWSSiteWiseAssetModel,
    displayCreateAWSIoTSiteWiseAssetModel,
    pressEnter,
    waitThirtySeconds,
    sdkCreateAWSIoTSiteWiseAssetModel,
    pressEnter,
    displayRetrievePropertyId,
    sdkRetrievePropertyId,
    pressEnter,
    displaySendDataToIoTSiteWiseAsset,
    sdkSendDataToIoTSiteWiseAsset,
    pressEnter,
```

```

    displayRetrieveValueOfIoTSiteWiseAsset,
    sdkRetrieveValueOfIoTSiteWiseAsset,
    pressEnter,
    displayCreateIoTSiteWisePortal,
    sdkCreateIoTSiteWisePortal,
    pressEnter,
    displayDescribePortal,
    sdkDescribePortal,
    pressEnter,
    displayCreateIoTSiteWiseGateway,
    sdkCreateIoTSiteWiseGateway,
    pressEnter,
    displayDescribeIoTSiteWiseGateway,
    sdkDescribeIoTSiteWiseGateway,
    pressEnter,
    askToDeleteResources,
    displayConfirmDeleteResources,
    sdkDeleteResources,
    goodbye,
  ],
  {
    iotSiteWiseClient: new IoTSiteWiseClient({}),
    cloudFormationClient: new CloudFormationClient({}),
    asset: { assetName: "MyAsset1" },
    assetModel: { assetModelName: "MyAssetModel1" },
    portal: { portalName: "MyPortal1" },
    gateway: { gatewayName: "MyGateway1" },
    propertyIds: [],
    contactEmail: "user@mydomain.com",
    thing: "MyThing1",
    sampleData: { temperature: 23.5, humidity: 65.0 },
  },
);

/** @type {{ stepHandlerOptions: StepHandlerOptions }} */
export const main = async (stepHandlerOptions) => {
  await myScenario.run(stepHandlerOptions);
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const { values } = parseArgs({
    options: {
      yes: {

```

```
        type: "boolean",
        short: "y",
      },
    },
  });
  main({ confirmAll: values.yes });
}
```

Aktionen

BatchPutAssetPropertyValue

Das folgende Codebeispiel zeigt die Verwendung `BatchPutAssetPropertyValue`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import {
  BatchPutAssetPropertyValueCommand,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * Batch put asset property values.
 * @param {{ entries : array }}
 */
export const main = async ({ entries }) => {
  const client = new IoTSiteWiseClient({});
  try {
    const result = await client.send(
      new BatchPutAssetPropertyValueCommand({
        entries: entries,
      }),
    );
  }
}
```

```

    );
    console.log("Asset properties batch put successfully.");
    return result;
  } catch (caught) {
    if (caught instanceof Error && caught.name === "ResourceNotFound") {
      console.warn(`${caught.message}. A resource could not be found.`);
    } else {
      throw caught;
    }
  }
};

```

- Einzelheiten zur API finden Sie [BatchPutAssetPropertyValue](#) in der AWS SDK für JavaScript API-Referenz.

CreateAsset

Das folgende Codebeispiel zeigt die Verwendung `CreateAsset`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```

import {
  CreateAssetCommand,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * Create an Asset.
 * @param {{ assetName : string, assetModelId: string }}
 */
export const main = async ({ assetName, assetModelId }) => {
  const client = new IoTSiteWiseClient({});
  try {

```

```
const result = await client.send(  
  new CreateAssetCommand({  
    assetName: assetName, // The name to give the Asset.  
    assetModelId: assetModelId, // The ID of the asset model from which to  
create the asset.  
  }),  
);  
console.log("Asset created successfully.");  
return result;  
} catch (caught) {  
  if (caught instanceof Error && caught.name === "ResourceNotFound") {  
    console.warn(  
      `${caught.message}. The asset model could not be found. Please check the  
asset model id.`,  
    );  
  } else {  
    throw caught;  
  }  
}  
};
```

- Einzelheiten zur API finden Sie [CreateAsset](#) in der AWS SDK für JavaScript API-Referenz.

CreateAssetModel

Das folgende Codebeispiel zeigt die Verwendung `CreateAssetModel`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import {  
  CreateAssetModelCommand,  
  IoTSiteWiseClient,  
} from "@aws-sdk/client-iotsitewise";  
import { parseArgs } from "node:util";
```

```
/**
 * Create an Asset Model.
 * @param {{ assetName : string, assetModelId: string }}
 */
export const main = async ({ assetModelName, assetModelId }) => {
  const client = new IoTSiteWiseClient({});
  try {
    const result = await client.send(
      new CreateAssetModelCommand({
        assetModelName: assetModelName, // The name to give the Asset Model.
      }),
    );
    console.log("Asset model created successfully.");
    return result;
  } catch (caught) {
    if (caught instanceof Error && caught.name === "IoTSiteWiseError") {
      console.warn(
        `${caught.message}. There was a problem creating the asset model.`
      );
    } else {
      throw caught;
    }
  }
};
```

- Einzelheiten zur API finden Sie [CreateAssetModel](#) in der AWS SDK für JavaScript API-Referenz.

CreateGateway

Das folgende Codebeispiel zeigt die Verwendung `CreateGateway`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import {
  CreateGatewayCommand,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * Create a Gateway.
 * @param {{ }}
 */
export const main = async ({ gatewayName }) => {
  const client = new IoTSiteWiseClient({});
  try {
    const result = await client.send(
      new CreateGatewayCommand({
        gatewayName: gatewayName, // The name to give the created Gateway.
      }),
    );
    console.log("Gateway created successfully.");
    return result;
  } catch (caught) {
    if (caught instanceof Error && caught.name === "IoTSiteWiseError") {
      console.warn(
        `${caught.message}. There was a problem creating the Gateway.`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Einzelheiten zur API finden Sie [CreateGateway](#) in der AWS SDK für JavaScript API-Referenz.

DeleteAsset

Das folgende Codebeispiel zeigt die Verwendung `DeleteAsset`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import {
  DeleteAssetCommand,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * Delete an asset.
 * @param {{ assetId : string }}
 */
export const main = async ({ assetId }) => {
  const client = new IoTSiteWiseClient({});
  try {
    await client.send(
      new DeleteAssetCommand({
        assetId: assetId, // The model id to delete.
      }),
    );
    console.log("Asset deleted successfully.");
    return { assetDeleted: true };
  } catch (caught) {
    if (caught instanceof Error && caught.name === "ResourceNotFound") {
      console.warn(
        `${caught.message}. There was a problem deleting the asset.`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Einzelheiten zur API finden Sie [DeleteAsset](#) in der AWS SDK für JavaScript API-Referenz.

DeleteAssetModel

Das folgende Codebeispiel zeigt die Verwendung `DeleteAssetModel`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import {
  DeleteAssetModelCommand,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * Delete an asset model.
 * @param {{ assetModelId : string }}
 */
export const main = async ({ assetModelId }) => {
  const client = new IoTSiteWiseClient({});
  try {
    await client.send(
      new DeleteAssetModelCommand({
        assetModelId: assetModelId, // The model id to delete.
      }),
    );
    console.log("Asset model deleted successfully.");
    return { assetModelDeleted: true };
  } catch (caught) {
    if (caught instanceof Error && caught.name === "ResourceNotFound") {
      console.warn(
        `${caught.message}. There was a problem deleting the asset model.`
      );
    } else {
      throw caught;
    }
  }
};
```

- Einzelheiten zur API finden Sie [DeleteAssetModel](#) in der AWS SDK für JavaScript API-Referenz.

DeleteGateway

Das folgende Codebeispiel zeigt die Verwendung `DeleteGateway`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import {
  DeleteGatewayCommand,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * Create an SSM document.
 * @param {{ content: string, name: string, documentType?: DocumentType }}
 */
export const main = async ({ gatewayId }) => {
  const client = new IoTSiteWiseClient({});
  try {
    await client.send(
      new DeleteGatewayCommand({
        gatewayId: gatewayId, // The ID of the Gateway to describe.
      }),
    );
    console.log("Gateway deleted successfully.");
    return { gatewayDeleted: true };
  } catch (caught) {
    if (caught instanceof Error && caught.name === "ResourceNotFound") {
      console.warn(
        `${caught.message}. The Gateway could not be found. Please check the Gateway Id.`
      );
    }
  }
};
```

```
    } else {  
      throw caught;  
    }  
  }  
};
```

- Einzelheiten zur API finden Sie [DeleteGateway](#) in der AWS SDK für JavaScript API-Referenz.

DescribeAssetModel

Das folgende Codebeispiel zeigt die Verwendung `DescribeAssetModel`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import {  
  DescribeAssetModelCommand,  
  IoTSiteWiseClient,  
} from "@aws-sdk/client-iotsitewise";  
import { parseArgs } from "node:util";  
  
/**  
 * Describe an asset model.  
 * @param {{ assetModelId : string }}  
 */  
export const main = async ({ assetModelId }) => {  
  const client = new IoTSiteWiseClient({});  
  try {  
    const { assetModelDescription } = await client.send(  
      new DescribeAssetModelCommand({  
        assetModelId: assetModelId, // The ID of the Gateway to describe.  
      }),  
    );  
    console.log("Asset model information retrieved successfully.");  
    return { assetModelDescription: assetModelDescription };  
  } catch (caught) {
```

```

    if (caught instanceof Error && caught.name === "ResourceNotFound") {
      console.warn(
        `${caught.message}. The asset model could not be found. Please check the
        asset model id.`
      );
    } else {
      throw caught;
    }
  }
};

```

- Einzelheiten zur API finden Sie [DescribeAssetModel](#) in der AWS SDK für JavaScript API-Referenz.

DescribeGateway

Das folgende Codebeispiel zeigt die Verwendung `DescribeGateway`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```

import {
  DescribeGatewayCommand,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * Create an SSM document.
 * @param {{ content: string, name: string, documentType?: DocumentType }}
 */
export const main = async ({ gatewayId }) => {
  const client = new IoTSiteWiseClient({});
  try {
    const { gatewayDescription } = await client.send(

```

```

    new DescribeGatewayCommand({
      gatewayId: gatewayId, // The ID of the Gateway to describe.
    }),
  );
  console.log("Gateway information retrieved successfully.");
  return { gatewayDescription: gatewayDescription };
} catch (caught) {
  if (caught instanceof Error && caught.name === "ResourceNotFound") {
    console.warn(
      `${caught.message}. The Gateway could not be found. Please check the Gateway
      Id.`
    );
  } else {
    throw caught;
  }
}
};

```

- Einzelheiten zur API finden Sie [DescribeGateway](#) in der AWS SDK für JavaScript API-Referenz.

GetAssetPropertyValue

Das folgende Codebeispiel zeigt die Verwendung `GetAssetPropertyValue`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```

import {
  GetAssetPropertyValueCommand,
  IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * Describe an asset property value.
 * @param {{ entryId : string }}

```

```
*/
export const main = async ({ entryId }) => {
  const client = new IoTSiteWiseClient({});
  try {
    const result = await client.send(
      new GetAssetPropertyValueCommand({
        entryId: entryId, // The ID of the Gateway to describe.
      }),
    );
    console.log("Asset property information retrieved successfully.");
    return result;
  } catch (caught) {
    if (caught instanceof Error && caught.name === "ResourceNotFound") {
      console.warn(
        `${caught.message}. The asset property entry could not be found. Please
check the entry id.`
      );
    } else {
      throw caught;
    }
  }
};
```

- Einzelheiten zur API finden Sie [GetAssetPropertyValue](#) in der AWS SDK für JavaScript API-Referenz.

ListAssetModels

Das folgende Codebeispiel zeigt die Verwendung `ListAssetModels`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import {
  ListAssetModelsCommand,
```

```
IoTSiteWiseClient,
} from "@aws-sdk/client-iotsitewise";
import { parseArgs } from "node:util";

/**
 * List asset models.
 * @param {{ assetModelTypes : array }}
 */
export const main = async ({ assetModelTypes = [] }) => {
  const client = new IoTSiteWiseClient({});
  try {
    const result = await client.send(
      new ListAssetModelsCommand({
        assetModelTypes: assetModelTypes, // The model types to list
      }),
    );
    console.log("Asset model types retrieved successfully.");
    return result;
  } catch (caught) {
    if (caught instanceof Error && caught.name === "IoTSiteWiseError") {
      console.warn(
        `${caught.message}. There was a problem listing the asset model types.`
      );
    } else {
      throw caught;
    }
  }
};
```

- Einzelheiten zur API finden Sie [ListAssetModels](#) in der AWS SDK für JavaScript API-Referenz.

Kinesis-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK für JavaScript (v3) mit Kinesis Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kodex finden.

Themen

- [Aktionen](#)
- [Serverless-Beispiele](#)

Aktionen

PutRecords

Das folgende Codebeispiel zeigt die Verwendung PutRecords von.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { PutRecordsCommand, KinesisClient } from "@aws-sdk/client-kinesis";

/**
 * Put multiple records into a Kinesis stream.
 * @param {{ streamArn: string }} config
 */
export const main = async ({ streamArn }) => {
  const client = new KinesisClient({});
  try {
    await client.send(
      new PutRecordsCommand({
        StreamARN: streamArn,
        Records: [
          {
            Data: new Uint8Array(),
            /**
             * Determines which shard in the stream the data record is assigned to.
             * Partition keys are Unicode strings with a maximum length limit of 256
            */
          }
        ]
      })
    );
  } catch (err) {
    console.error(err);
  }
}
```



```

        * characters for each key. Amazon Kinesis Data Streams uses the
partition
        * key as input to a hash function that maps the partition key and
        * associated data to a specific shard.
        */
        PartitionKey: "TEST_KEY",
    },
    {
        Data: new Uint8Array(),
        PartitionKey: "TEST_KEY",
    },
  ],
 )),
);
} catch (caught) {
  if (caught instanceof Error) {
    //
  } else {
    throw caught;
  }
}
};

// Call function if run directly.
import { fileURLToPath } from "node:url";
import { parseArgs } from "node:util";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const options = {
    streamArn: {
      type: "string",
      description: "The ARN of the stream.",
    },
  };

  const { values } = parseArgs({ options });
  main(values);
}

```

- Einzelheiten zur API finden Sie [PutRecords](#) in der AWS SDK für JavaScript API-Referenz.

Serverless-Beispiele

Aufrufen einer Lambda-Funktion über einen Kinesis-Auslöser

Das folgende Codebeispiel veranschaulicht, wie eine Lambda-Funktion implementiert wird, die ein durch den Empfang von Datensätzen aus einem Kinesis-Stream ausgelöstes Ereignis empfängt. Die Funktion ruft die Kinesis-Nutzlast ab, dekodiert von Base64 und protokolliert den Datensatzinhalt.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Ein Kinesis-Ereignis mit Lambda verwenden. JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    try {
      console.log(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      console.log(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      console.error(`An error occurred ${err}`);
      throw err;
    }
  }
  console.log(`Successfully processed ${event.Records.length} records.`);
};

async function getRecordDataAsync(payload) {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

Ein Kinesis-Ereignis mit Lambda verwenden. TypeScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  KinesisStreamEvent,
  Context,
  KinesisStreamHandler,
  KinesisStreamRecordPayload,
} from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";

const logger = new Logger({
  logLevel: "INFO",
  serviceName: "kinesis-stream-handler-sample",
});

export const functionHandler: KinesisStreamHandler = async (
  event: KinesisStreamEvent,
  context: Context
): Promise<void> => {
  for (const record of event.Records) {
    try {
      logger.info(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      logger.info(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      logger.error(`An error occurred ${err}`);
      throw err;
    }
    logger.info(`Successfully processed ${event.Records.length} records.`);
  }
};

async function getRecordDataAsync(
  payload: KinesisStreamRecordPayload
): Promise<string> {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

Melden von Batch-Elementfehlern für Lambda-Funktionen mit einem Kinesis-Auslöser

Das folgende Codebeispiel zeigt, wie eine teilweise Batch-Antwort für Lambda-Funktionen implementiert wird, die Ereignisse von einem Kinesis-Stream empfangen. Die Funktion meldet die Batch-Elementfehler in der Antwort und signalisiert Lambda, diese Nachrichten später erneut zu versuchen.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Melden von Fehlern bei Kinesis-Batchelementen mit Lambda unter Verwendung von Javascript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    try {
      console.log(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      console.log(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      console.error(`An error occurred ${err}`);
      /* Since we are working with streams, we can return the failed item
      immediately.
      Lambda will immediately begin to retry processing from this failed item
      onwards. */
      return {
        batchItemFailures: [{ itemIdentifier: record.kinesis.sequenceNumber }],
      };
    }
  }
  console.log(`Successfully processed ${event.Records.length} records.`);
  return { batchItemFailures: [] };
}
```

```
};

async function getRecordDataAsync(payload) {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

Melden von Fehlern Kinesis Kinesis-Batch-Elementen mit Lambda unter Verwendung von TypeScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  KinesisStreamEvent,
  Context,
  KinesisStreamHandler,
  KinesisStreamRecordPayload,
  KinesisStreamBatchResponse,
} from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";

const logger = new Logger({
  logLevel: "INFO",
  serviceName: "kinesis-stream-handler-sample",
});

export const functionHandler: KinesisStreamHandler = async (
  event: KinesisStreamEvent,
  context: Context
): Promise<KinesisStreamBatchResponse> => {
  for (const record of event.Records) {
    try {
      logger.info(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      logger.info(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      logger.error(`An error occurred ${err}`);
      /* Since we are working with streams, we can return the failed item
      immediately.
```

```
        Lambda will immediately begin to retry processing from this failed item
onwards. */
    return {
        batchItemFailures: [{ itemIdentifier: record.kinesis.sequenceNumber }],
    };
}
}
logger.info(`Successfully processed ${event.Records.length} records.`);
return { batchItemFailures: [] };
};

async function getRecordDataAsync(
    payload: KinesisStreamRecordPayload
): Promise<string> {
    var data = Buffer.from(payload.data, "base64").toString("utf-8");
    await Promise.resolve(1); //Placeholder for actual async work
    return data;
}
```

Lambda-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie AWS SDK für JavaScript (v3) mit Lambda verwenden.

Bei Grundlagen handelt es sich um Codebeispiele, die Ihnen zeigen, wie Sie die wesentlichen Vorgänge innerhalb eines Services ausführen.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien anzeigen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb eines Service aufrufen oder mit anderen AWS-Services kombinieren.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kodex finden.

Themen

- [Erste Schritte](#)

- [Grundlagen](#)
- [Aktionen](#)
- [Szenarien](#)
- [Serverless-Beispiele](#)

Erste Schritte

Hallo Lambda

Das folgende Codebeispiel zeigt, wie Sie mit Lambda beginnen können.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { LambdaClient, paginateListFunctions } from "@aws-sdk/client-lambda";

const client = new LambdaClient({});

export const helloLambda = async () => {
  const paginator = paginateListFunctions({ client }, {});
  const functions = [];

  for await (const page of paginator) {
    const funcNames = page.Functions.map((f) => f.FunctionName);
    functions.push(...funcNames);
  }

  console.log("Functions:");
  console.log(functions.join("\n"));
  return functions;
};
```

- Einzelheiten zur API finden Sie [ListFunctions](#) in der AWS SDK für JavaScript API-Referenz.

Grundlagen

Kennenlernen der Grundlagen

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Erstellen Sie eine IAM-Rolle und eine Lambda-Funktion und laden Sie den Handlercode hoch.
- Rufen Sie die Funktion mit einem einzigen Parameter auf und erhalten Sie Ergebnisse.
- Aktualisieren Sie den Funktionscode und konfigurieren Sie mit einer Umgebungsvariablen.
- Rufen Sie die Funktion mit neuen Parametern auf und erhalten Sie Ergebnisse. Zeigt das zurückgegebene Ausführungsprotokoll an.
- Listen Sie die Funktionen für Ihr Konto auf und bereinigen Sie dann die Ressourcen.

Weitere Informationen zur Verwendung von Lambda finden Sie unter [Erstellen einer Lambda-Funktion mit der Konsole](#).

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie eine AWS Identity and Access Management (IAM-) Rolle, die Lambda die Berechtigung erteilt, in Protokolle zu schreiben.

```
logger.log(`Creating role (${NAME_ROLE_LAMBDA})...`);
const response = await createRole(NAME_ROLE_LAMBDA);

import { AttachRolePolicyCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} policyArn
 * @param {string} roleName
```



```
*/  
export const attachRolePolicy = (policyArn, roleName) => {  
  const command = new AttachRolePolicyCommand({  
    PolicyArn: policyArn,  
    RoleName: roleName,  
  });  
  
  return client.send(command);  
};
```

Erstellen Sie eine Lambda-Funktion und laden Sie Handlercode hoch.

```
const createFunction = async (funcName, roleArn) => {  
  const client = new LambdaClient({});  
  const code = await readFile(`${dirname}../functions/${funcName}.zip`);  
  
  const command = new CreateFunctionCommand({  
    Code: { ZipFile: code },  
    FunctionName: funcName,  
    Role: roleArn,  
    Architectures: [Architecture.arm64],  
    Handler: "index.handler", // Required when sending a .zip file  
    PackageType: PackageType.Zip, // Required when sending a .zip file  
    Runtime: Runtime.nodejs16x, // Required when sending a .zip file  
  });  
  
  return client.send(command);  
};
```

Rufen Sie die Funktion mit einem einzigen Parameter auf und erhalten Sie Ergebnisse.

```
const invoke = async (funcName, payload) => {  
  const client = new LambdaClient({});  
  const command = new InvokeCommand({  
    FunctionName: funcName,  
    Payload: JSON.stringify(payload),  
    LogType: LogType.Tail,  
  });  
  
  const { Payload, LogResult } = await client.send(command);  
  const result = Buffer.from(Payload).toString();  
};
```

```
const logs = Buffer.from(LogResult, "base64").toString();
return { logs, result };
};
```

Aktualisieren Sie den Funktionscode und konfigurieren Sie seine Lambda-Umgebung mit einer Umgebungsvariablen.

```
const updateFunctionCode = async (funcName, newFunc) => {
  const client = new LambdaClient({});
  const code = await readFile(`${dirname}../functions/${newFunc}.zip`);
  const command = new UpdateFunctionCodeCommand({
    ZipFile: code,
    FunctionName: funcName,
    Architectures: [Architecture.arm64],
    Handler: "index.handler", // Required when sending a .zip file
    PackageType: PackageType.Zip, // Required when sending a .zip file
    Runtime: Runtime.nodejs16x, // Required when sending a .zip file
  });

  return client.send(command);
};

const updateFunctionConfiguration = (funcName) => {
  const client = new LambdaClient({});
  const config = readFileSync(`${dirname}../functions/config.json`).toString();
  const command = new UpdateFunctionConfigurationCommand({
    ...JSON.parse(config),
    FunctionName: funcName,
  });
  const result = client.send(command);
  waitForFunctionUpdated({ FunctionName: funcName });
  return result;
};
```

Listen Sie die Funktionen für Ihr Konto auf.

```
const listFunctions = () => {
  const client = new LambdaClient({});
  const command = new ListFunctionsCommand({});

  return client.send(command);
};
```

```
};
```

Löschen Sie die IAM-Rolle für Ihre Lambda-Funktion.

```
import { DeleteRoleCommand, IAMClient } from "@aws-sdk/client-iam";

const client = new IAMClient({});

/**
 *
 * @param {string} roleName
 */
export const deleteRole = (roleName) => {
  const command = new DeleteRoleCommand({ RoleName: roleName });
  return client.send(command);
};

/**
 * @param {string} funcName
 */
const deleteFunction = (funcName) => {
  const client = new LambdaClient({});
  const command = new DeleteFunctionCommand({ FunctionName: funcName });
  return client.send(command);
};
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für JavaScript - API-Referenz.
 - [CreateFunction](#)
 - [DeleteFunction](#)
 - [GetFunction](#)
 - [Invoke](#)
 - [ListFunctions](#)
 - [UpdateFunctionCode](#)
 - [UpdateFunctionConfiguration](#)

Aktionen

CreateFunction

Das folgende Codebeispiel zeigt, wie man es benutzt. CreateFunction

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
const createFunction = async (funcName, roleArn) => {
  const client = new LambdaClient({});
  const code = await readFile(`${dirname}../functions/${funcName}.zip`);

  const command = new CreateFunctionCommand({
    Code: { ZipFile: code },
    FunctionName: funcName,
    Role: roleArn,
    Architectures: [Architecture.arm64],
    Handler: "index.handler", // Required when sending a .zip file
    PackageType: PackageType.Zip, // Required when sending a .zip file
    Runtime: Runtime.nodejs16x, // Required when sending a .zip file
  });

  return client.send(command);
};
```

- Einzelheiten zur API finden Sie [CreateFunction](#) in der AWS SDK für JavaScript API-Referenz.

DeleteFunction

Das folgende Codebeispiel zeigt die Verwendung DeleteFunction.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
/**
 * @param {string} funcName
 */
const deleteFunction = (funcName) => {
  const client = new LambdaClient({});
  const command = new DeleteFunctionCommand({ FunctionName: funcName });
  return client.send(command);
};
```

- Einzelheiten zur API finden Sie [DeleteFunction](#) in der AWS SDK für JavaScript API-Referenz.

GetFunction

Das folgende Codebeispiel zeigt die Verwendung `GetFunction`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
const getFunction = (funcName) => {
  const client = new LambdaClient({});
  const command = new GetFunctionCommand({ FunctionName: funcName });
  return client.send(command);
};
```

- Einzelheiten zur API finden Sie [GetFunction](#) in der AWS SDK für JavaScript API-Referenz.

Invoke

Das folgende Codebeispiel zeigt die Verwendung `Invoke`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
const invoke = async (funcName, payload) => {
  const client = new LambdaClient({});
  const command = new InvokeCommand({
    FunctionName: funcName,
    Payload: JSON.stringify(payload),
    LogType: LogType.Tail,
  });

  const { Payload, LogResult } = await client.send(command);
  const result = Buffer.from(Payload).toString();
  const logs = Buffer.from(LogResult, "base64").toString();
  return { logs, result };
};
```

- Weitere API-Informationen finden Sie unter [Invoke](#) in der AWS SDK für JavaScript -API-Referenz.

ListFunctions

Das folgende Codebeispiel zeigt die Verwendung `ListFunctions`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
const listFunctions = () => {  
  const client = new LambdaClient({});  
  const command = new ListFunctionsCommand({});  
  
  return client.send(command);  
};
```

- Einzelheiten zur API finden Sie [ListFunctions](#) in der AWS SDK für JavaScript API-Referenz.

UpdateFunctionCode

Das folgende Codebeispiel zeigt die Verwendung `UpdateFunctionCode`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
const updateFunctionCode = async (funcName, newFunc) => {  
  const client = new LambdaClient({});  
  const code = await readFile(`${dirname}../functions/${newFunc}.zip`);  
  const command = new UpdateFunctionCodeCommand({  
    ZipFile: code,  
    FunctionName: funcName,  
    Architectures: [Architecture.arm64],  
    Handler: "index.handler", // Required when sending a .zip file  
    PackageType: PackageType.Zip, // Required when sending a .zip file  
    Runtime: Runtime.nodejs16x, // Required when sending a .zip file  
  });  
  
  return client.send(command);  
};
```

- Einzelheiten zur API finden Sie [UpdateFunctionCode](#) in der AWS SDK für JavaScript API-Referenz.

UpdateFunctionConfiguration

Das folgende Codebeispiel zeigt die Verwendung `UpdateFunctionConfiguration`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
const updateFunctionConfiguration = (funcName) => {
  const client = new LambdaClient({});
  const config = readFileSync(`${dirname}../functions/config.json`).toString();
  const command = new UpdateFunctionConfigurationCommand({
    ...JSON.parse(config),
    FunctionName: funcName,
  });
  const result = client.send(command);
  waitForFunctionUpdated({ FunctionName: funcName });
  return result;
};
```

- Einzelheiten zur API finden Sie [UpdateFunctionConfiguration](#) in der AWS SDK für JavaScript API-Referenz.

Szenarien

Bestätigen Sie automatisch bekannte Benutzer mit einer Lambda-Funktion

Die folgenden Codebeispiele zeigen, wie bekannte Amazon-Cognito-Benutzer mit einer Lambda-Funktion automatisch bestätigt werden können.

- Konfigurieren Sie einen Benutzerpool, um eine Lambda-Funktion für den `PreSignUp`-Trigger aufzurufen.
- Registrieren eines Benutzers bei Amazon Cognito.
- Die Lambda-Funktion scannt eine DynamoDB-Tabelle und bestätigt automatisch bekannte Benutzer.

- Melden Sie sich als neuer Benutzer an und bereinigen Sie dann die Ressourcen.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Konfigurieren Sie einen interaktiven „Szenario“-Lauf. Die JavaScript (v3) -Beispiele verwenden gemeinsam einen Szenario-Runner, um komplexe Beispiele zu vereinfachen. Der komplette Quellcode ist aktiviert GitHub.

```
import { AutoConfirm } from "../scenario-auto-confirm.js";

/**
 * The context is passed to every scenario. Scenario steps
 * will modify the context.
 */
const context = {
  errors: [],
  users: [
    {
      UserName: "test_user_1",
      userEmail: "test_email_1@example.com",
    },
    {
      UserName: "test_user_2",
      userEmail: "test_email_2@example.com",
    },
    {
      UserName: "test_user_3",
      userEmail: "test_email_3@example.com",
    },
  ],
};

/**
 * Three Scenarios are created for the workflow. A Scenario is an orchestration
 * class
```

```

    * that simplifies running a series of steps.
    */
export const scenarios = {
    // Demonstrate automatically confirming known users in a database.
    "auto-confirm": AutoConfirm(context),
};

// Call function if run directly
import { fileURLToPath } from "node:url";
import { parseScenarioArgs } from "@aws-doc-sdk-examples/lib/scenario/index.js";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
    parseScenarioArgs(scenarios, {
        name: "Cognito user pools and triggers",
        description:
            "Demonstrate how to use the AWS SDKs to customize Amazon Cognito authentication behavior.",
    });
}

```

Dieses Szenario demonstriert die automatische Bestätigung eines bekannten Benutzers. Es orchestriert die Beispielschritte.

```

import { wait } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";
import {
    Scenario,
    ScenarioAction,
    ScenarioInput,
    ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/scenario.js";

import {
    getStackOutputs,
    logCleanUpReminder,
    promptForStackName,
    promptForStackRegion,
    skipWhenErrors,
} from "../steps-common.js";
import { populateTable } from "../actions/dynamodb-actions.js";
import {
    addPreSignUpHandler,
    deleteUser,

```

```

    getUser,
    signIn,
    signUpUser,
  } from "./actions/cognito-actions.js";
import {
  getLatestLogStreamForLambda,
  getLogEvents,
} from "./actions/cloudwatch-logs-actions.js";

/**
 * @typedef {{
 *   errors: Error[],
 *   password: string,
 *   users: { UserName: string, UserEmail: string }[],
 *   selectedUser?: string,
 *   stackName?: string,
 *   stackRegion?: string,
 *   token?: string,
 *   confirmDeleteSignedInUser?: boolean,
 *   TableName?: string,
 *   UserPoolClientId?: string,
 *   UserPoolId?: string,
 *   UserPoolArn?: string,
 *   AutoConfirmHandlerArn?: string,
 *   AutoConfirmHandlerName?: string
 * }} State
 */

const greeting = new ScenarioOutput(
  "greeting",
  (/** @type {State} */ state) => `This demo will populate some users into the \
database created as part of the "${state.stackName}" stack. \
Then the AutoConfirmHandler will be linked to the PreSignUp \
trigger from Cognito. Finally, you will choose a user to sign up.`,
  { skipWhen: skipWhenErrors },
);

const logPopulatingUsers = new ScenarioOutput(
  "logPopulatingUsers",
  "Populating the DynamoDB table with some users.",
  { skipWhenErrors: skipWhenErrors },
);

const logPopulatingUsersComplete = new ScenarioOutput(

```

```
    "logPopulatingUsersComplete",
    "Done populating users.",
    { skipWhen: skipWhenErrors },
  );

const populateUsers = new ScenarioAction(
  "populateUsers",
  async (** @type {State} */ state) => {
    const [, err] = await populateTable({
      region: state.stackRegion,
      tableName: state.TableName,
      items: state.users,
    });
    if (err) {
      state.errors.push(err);
    }
  },
  {
    skipWhen: skipWhenErrors,
  },
);

const logSetupSignUpTrigger = new ScenarioOutput(
  "logSetupSignUpTrigger",
  "Setting up the PreSignUp trigger for the Cognito User Pool.",
  { skipWhen: skipWhenErrors },
);

const setupSignUpTrigger = new ScenarioAction(
  "setupSignUpTrigger",
  async (** @type {State} */ state) => {
    const [, err] = await addPreSignUpHandler({
      region: state.stackRegion,
      userPoolId: state.UserPoolId,
      handlerArn: state.AutoConfirmHandlerArn,
    });
    if (err) {
      state.errors.push(err);
    }
  },
  {
    skipWhen: skipWhenErrors,
  },
);
```

```

const logSetupSignUpTriggerComplete = new ScenarioOutput(
  "logSetupSignUpTriggerComplete",
  (
    /** @type {State} */ state,
  ) => `The lambda function "${state.AutoConfirmHandlerName}" \
has been configured as the PreSignUp trigger handler for the user pool
"${state.UserPoolId}".`,
  { skipWhen: skipWhenErrors },
);

const selectUser = new ScenarioInput(
  "selectedUser",
  "Select a user to sign up.",
  {
    type: "select",
    choices: (/** @type {State} */ state) => state.users.map((u) => u.UserName),
    skipWhen: skipWhenErrors,
    default: (/** @type {State} */ state) => state.users[0].UserName,
  },
);

const checkIfUserAlreadyExists = new ScenarioAction(
  "checkIfUserAlreadyExists",
  async (/** @type {State} */ state) => {
    const [user, err] = await getUser({
      region: state.stackRegion,
      userPoolId: state.UserPoolId,
      username: state.selectedUser,
    });

    if (err?.name === "UserNotFoundException") {
      // Do nothing. We're not expecting the user to exist before
      // sign up is complete.
      return;
    }

    if (err) {
      state.errors.push(err);
      return;
    }

    if (user) {
      state.errors.push(

```

```
        new Error(
            `The user "${state.selectedUser}" already exists in the user pool
"${state.UserPoolId}".`,
        ),
    );
}
},
{
    skipWhen: skipWhenErrors,
},
);

const createPassword = new ScenarioInput(
    "password",
    "Enter a password that has at least eight characters, uppercase, lowercase,
numbers and symbols.",
    { type: "password", skipWhen: skipWhenErrors, default: "Abcd1234!" },
);

const logSignUpExistingUser = new ScenarioOutput(
    "logSignUpExistingUser",
    (/** @type {State} */ state) => `Signing up user "${state.selectedUser}".`,
    { skipWhen: skipWhenErrors },
);

const signUpExistingUser = new ScenarioAction(
    "signUpExistingUser",
    async (/** @type {State} */ state) => {
        const signUp = (password) =>
            signUpUser({
                region: state.stackRegion,
                userPoolClientId: state.UserPoolClientId,
                username: state.selectedUser,
                email: state.users.find((u) => u.UserName === state.selectedUser)
                    .UserEmail,
                password,
            });

        let [, err] = await signUp(state.password);

        while (err?.name === "InvalidPasswordException") {
            console.warn("The password you entered was invalid.");
            await createPassword.handle(state);
            [, err] = await signUp(state.password);
        }
    }
);
```

```
    }

    if (err) {
      state.errors.push(err);
    }
  },
  { skipWhen: skipWhenErrors },
);

const logSignUpExistingUserComplete = new ScenarioOutput(
  "logSignUpExistingUserComplete",
  (** @type {State} */ state) =>
    `${state.selectedUser} was signed up successfully.`,
  { skipWhen: skipWhenErrors },
);

const logLambdaLogs = new ScenarioAction(
  "logLambdaLogs",
  async (** @type {State} */ state) => {
    console.log(
      "Waiting a few seconds to let Lambda write to CloudWatch Logs...\n",
    );
    await wait(10);

    const [logStream, logStreamErr] = await getLatestLogStreamForLambda({
      functionName: state.AutoConfirmHandlerName,
      region: state.stackRegion,
    });
    if (logStreamErr) {
      state.errors.push(logStreamErr);
      return;
    }

    console.log(
      `Getting some recent events from log stream "${logStream.logStreamName}"`,
    );
    const [logEvents, logEventsErr] = await getLogEvents({
      functionName: state.AutoConfirmHandlerName,
      region: state.stackRegion,
      eventCount: 10,
      logStreamName: logStream.logStreamName,
    });
    if (logEventsErr) {
      state.errors.push(logEventsErr);
    }
  }
);
```

```
        return;
    }

    console.log(logEvents.map((ev) => `\t${ev.message}`).join(""));
},
{ skipWhen: skipWhenErrors },
);

const logSignInUser = new ScenarioOutput(
    "logSignInUser",
    (/** @type {State} */ state) => `Let's sign in as ${state.selectedUser}`,
    { skipWhen: skipWhenErrors },
);

const signInUser = new ScenarioAction(
    "signInUser",
    async (/** @type {State} */ state) => {
        const [response, err] = await signIn({
            region: state.stackRegion,
            clientId: state.UserPoolClientId,
            username: state.selectedUser,
            password: state.password,
        });

        if (err?.name === "PasswordResetRequiredException") {
            state.errors.push(new Error("Please reset your password."));
            return;
        }

        if (err) {
            state.errors.push(err);
            return;
        }

        state.token = response?.AuthenticationResult?.AccessToken;
    },
    { skipWhen: skipWhenErrors },
);

const logSignInUserComplete = new ScenarioOutput(
    "logSignInUserComplete",
    (/** @type {State} */ state) =>
        `Successfully signed in. Your access token starts with: ${state.token.slice(0, 11)}`,
);
```



```
    { skipWhen: skipWhenErrors },
  );

const confirmDeleteSignedInUser = new ScenarioInput(
  "confirmDeleteSignedInUser",
  "Do you want to delete the currently signed in user?",
  { type: "confirm", skipWhen: skipWhenErrors },
);

const deleteSignedInUser = new ScenarioAction(
  "deleteSignedInUser",
  async (** @type {State} */ state) => {
    const [, err] = await deleteUser({
      region: state.stackRegion,
      accessToken: state.token,
    });

    if (err) {
      state.errors.push(err);
    }
  },
  {
    skipWhen: (** @type {State} */ state) =>
      skipWhenErrors(state) || !state.confirmDeleteSignedInUser,
  },
);

const logErrors = new ScenarioOutput(
  "logErrors",
  (** @type {State} */ state) => {
    const errorList = state.errors
      .map((err) => ` - ${err.name}: ${err.message}`)
      .join("\n");
    return `Scenario errors found:\n${errorList}`;
  },
  {
    // Don't log errors when there aren't any!
    skipWhen: (** @type {State} */ state) => state.errors.length === 0,
  },
);

export const AutoConfirm = (context) =>
  new Scenario(
    "AutoConfirm",
```

```
[
  promptForStackName,
  promptForStackRegion,
  getStackOutputs,
  greeting,
  logPopulatingUsers,
  populateUsers,
  logPopulatingUsersComplete,
  logSetupSignUpTrigger,
  setupSignUpTrigger,
  logSetupSignUpTriggerComplete,
  selectUser,
  checkIfUserAlreadyExists,
  createPassword,
  logSignUpExistingUser,
  signUpExistingUser,
  logSignUpExistingUserComplete,
  logLambdaLogs,
  logSignInUser,
  signInUser,
  logSignInUserComplete,
  confirmDeleteSignedInUser,
  deleteSignedInUser,
  logCleanUpReminder,
  logErrors,
],
context,
);
```

Dies sind Schritte, die mit anderen Szenarien geteilt werden.

```
import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/scenario.js";
import { getCfnOutputs } from "@aws-doc-sdk-examples/lib/sdk/cfn-outputs.js";

export const skipWhenErrors = (state) => state.errors.length > 0;

export const getStackOutputs = new ScenarioAction(
  "getStackOutputs",
```

```

    async (state) => {
      if (!state.stackName || !state.stackRegion) {
        state.errors.push(
          new Error(
            "No stack name or region provided. The stack name and \
region are required to fetch CFN outputs relevant to this example.",
          ),
        );
        return;
      }

      const outputs = await getCfnOutputs(state.stackName, state.stackRegion);
      Object.assign(state, outputs);
    },
  );

  export const promptForStackName = new ScenarioInput(
    "stackName",
    "Enter the name of the stack you deployed earlier.",
    { type: "input", default: "PoolsAndTriggersStack" },
  );

  export const promptForStackRegion = new ScenarioInput(
    "stackRegion",
    "Enter the region of the stack you deployed earlier.",
    { type: "input", default: "us-east-1" },
  );

  export const logCleanUpReminder = new ScenarioOutput(
    "logCleanUpReminder",
    "All done. Remember to run 'cdk destroy' to teardown the stack.",
    { skipWhen: skipWhenErrors },
  );

```

Ein Handler für den PreSignUp-Trigger mit einer Lambda-Funktion.

```

import type { PreSignUpTriggerEvent, Handler } from "aws-lambda";
import type { UserRepository } from "../user-repository";
import { DynamoDBUserRepository } from "../user-repository";

export class PreSignUpHandler {
  private userRepository: UserRepository;

```

```
constructor(userRepository: UserRepository) {
  this.userRepository = userRepository;
}

private isPreSignUpTriggerSource(event: PreSignUpTriggerEvent): boolean {
  return event.triggerSource === "PreSignUp_SignUp";
}

private getEventUserEmail(event: PreSignUpTriggerEvent): string {
  return event.request.userAttributes.email;
}

async handlePreSignUpTriggerEvent(
  event: PreSignUpTriggerEvent,
): Promise<PreSignUpTriggerEvent> {
  console.log(
    `Received presignup from ${event.triggerSource} for user '${event.userName}'`,
  );

  if (!this.isPreSignUpTriggerSource(event)) {
    return event;
  }

  const eventEmail = this.getEventUserEmail(event);
  console.log(`Looking up email ${eventEmail}.`);
  const storedUserInfo =
    await this.userRepository.getUserInfoByEmail(eventEmail);

  if (!storedUserInfo) {
    console.log(
      `Email ${eventEmail} not found. Email verification is required.`,
    );
    return event;
  }

  if (storedUserInfo.UserName !== event.userName) {
    console.log(
      `UserEmail ${eventEmail} found, but stored UserName
      '${storedUserInfo.UserName}' does not match supplied UserName '${event.userName}'.
      Verification is required.`,
    );
  } else {
    console.log(
```

```

        `UserEmail ${eventEmail} found with matching UserName
        ${storedUserInfo.UserName}. User is confirmed.` ,
        );
        event.response.autoConfirmUser = true;
        event.response.autoVerifyEmail = true;
    }
    return event;
}
}

const createPreSignUpHandler = (): PreSignUpHandler => {
    const tableName = process.env.TABLE_NAME;
    if (!tableName) {
        throw new Error("TABLE_NAME environment variable is not set");
    }

    const userRepository = new DynamoDBUserRepository(tableName);
    return new PreSignUpHandler(userRepository);
};

export const handler: Handler = async (event: PreSignUpTriggerEvent) => {
    const preSignUpHandler = createPreSignUpHandler();
    return preSignUpHandler.handlePreSignUpTriggerEvent(event);
};

```

Modul für CloudWatch Logs-Aktionen.

```

import {
    CloudWatchLogsClient,
    GetLogEventsCommand,
    OrderBy,
    paginateDescribeLogStreams,
} from "@aws-sdk/client-cloudwatch-logs";

/**
 * Get the latest log stream for a Lambda function.
 * @param {{ functionName: string, region: string }} config
 * @returns {Promise<[import("@aws-sdk/client-cloudwatch-logs").LogStream | null,
    unknown]>}
 */
export const getLatestLogStreamForLambda = async ({ functionName, region }) => {

```

```

    try {
      const logGroupName = `/aws/lambda/${functionName}`;
      const cwlClient = new CloudWatchLogsClient({ region });
      const paginator = paginateDescribeLogStreams(
        { client: cwlClient },
        {
          descending: true,
          limit: 1,
          orderBy: OrderBy.LastEventTime,
          logGroupName,
        },
      );

      for await (const page of paginator) {
        return [page.logStreams[0], null];
      }
    } catch (err) {
      return [null, err];
    }
  };

/**
 * Get the log events for a Lambda function's log stream.
 * @param {{
 *   functionName: string,
 *   logStreamName: string,
 *   eventCount: number,
 *   region: string
 * }} config
 * @returns {Promise<[import("@aws-sdk/client-cloudwatch-logs").OutputLogEvent[] |
 * null, unknown]>}
 */
export const getLogEvents = async ({
  functionName,
  logStreamName,
  eventCount,
  region,
}) => {
  try {
    const cwlClient = new CloudWatchLogsClient({ region });
    const logGroupName = `/aws/lambda/${functionName}`;
    const response = await cwlClient.send(
      new GetLogEventsCommand({
        logStreamName: logStreamName,

```

```

        limit: eventCount,
        logGroupName: logGroupName,
    )),
    );

    return [response.events, null];
} catch (err) {
    return [null, err];
}
};

```

Modul für Amazon-Cognito-Aktionen.

```

import {
    AdminGetUserCommand,
    CognitoIdentityProviderClient,
    DeleteUserCommand,
    InitiateAuthCommand,
    SignUpCommand,
    UpdateUserPoolCommand,
} from "@aws-sdk/client-cognito-identity-provider";

/**
 * Connect a Lambda function to the PreSignUp trigger for a Cognito user pool
 * @param {{ region: string, userPoolId: string, handlerArn: string }} config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-provider").UpdateUserPoolCommandOutput | null, unknown]>}
 */
export const addPreSignUpHandler = async ({
    region,
    userPoolId,
    handlerArn,
}) => {
    try {
        const cognitoClient = new CognitoIdentityProviderClient({
            region,
        });

        const command = new UpdateUserPoolCommand({
            UserPoolId: userPoolId,
            LambdaConfig: {

```

```
        PreSignUp: handlerArn,
      },
    });

    const response = await cognitoClient.send(command);
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};

/**
 * Attempt to register a user to a user pool with a given username and password.
 * @param {{
 *   region: string,
 *   userPoolClientId: string,
 *   username: string,
 *   email: string,
 *   password: string
 * }} config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-provider").SignUpCommandOutput | null, unknown]>}
 */
export const signUpUser = async ({
  region,
  userPoolClientId,
  username,
  email,
  password,
}) => {
  try {
    const cognitoClient = new CognitoIdentityProviderClient({
      region,
    });

    const response = await cognitoClient.send(
      new SignUpCommand({
        ClientId: userPoolClientId,
        Username: username,
        Password: password,
        UserAttributes: [{ Name: "email", Value: email }],
      }),
    );
    return [response, null];
  }
};
```



```
    } catch (err) {
      return [null, err];
    }
  };

/**
 * Sign in a user to Amazon Cognito using a username and password authentication
 * flow.
 * @param {{ region: string, clientId: string, username: string, password: string }}
 * config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-
 * provider").InitiateAuthCommandOutput | null, unknown]>}
 */
export const signIn = async ({ region, clientId, username, password }) => {
  try {
    const cognitoClient = new CognitoIdentityProviderClient({ region });
    const response = await cognitoClient.send(
      new InitiateAuthCommand({
        AuthFlow: "USER_PASSWORD_AUTH",
        ClientId: clientId,
        AuthParameters: { USERNAME: username, PASSWORD: password },
      }),
    );
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};

/**
 * Retrieve an existing user from a user pool.
 * @param {{ region: string, userPoolId: string, username: string }} config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-
 * provider").AdminGetUserCommandOutput | null, unknown]>}
 */
export const getUser = async ({ region, userPoolId, username }) => {
  try {
    const cognitoClient = new CognitoIdentityProviderClient({ region });
    const response = await cognitoClient.send(
      new AdminGetUserCommand({
        UserPoolId: userPoolId,
        Username: username,
      }),
    );
  }
};
```

```

        return [response, null];
    } catch (err) {
        return [null, err];
    }
};

/**
 * Delete the signed-in user. Useful for allowing a user to delete their
 * own profile.
 * @param {{ region: string, accessToken: string }} config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-provider").DeleteUserCommandOutput | null, unknown]>}
 */
export const deleteUser = async ({ region, accessToken }) => {
    try {
        const client = new CognitoIdentityProviderClient({ region });
        const response = await client.send(
            new DeleteUserCommand({ AccessToken: accessToken }),
        );
        return [response, null];
    } catch (err) {
        return [null, err];
    }
};

```

Modul für DynamoDB-Aktionen.

```

import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
    BatchWriteCommand,
    DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

/**
 * Populate a DynamoDB table with provide items.
 * @param {{ region: string, tableName: string, items: Record<string, unknown>[] }}
 * config
 * @returns {Promise<[import("@aws-sdk/lib-dynamodb").BatchWriteCommandOutput |
 * null, unknown]>}
 */
export const populateTable = async ({ region, tableName, items }) => {

```

```
try {
  const ddbClient = new DynamoDBClient({ region });
  const docClient = DynamoDBDocumentClient.from(ddbClient);
  const response = await docClient.send(
    new BatchWriteCommand({
      RequestItems: {
        [tableName]: items.map((item) => ({
          PutRequest: {
            Item: item,
          },
        })),
      },
    }),
  );
  return [response, null];
} catch (err) {
  return [null, err];
}
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für JavaScript - API-Referenz.
 - [DeleteUser](#)
 - [InitiateAuth](#)
 - [SignUp](#)
 - [UpdateUserPool](#)

Erstellen einer Serverless-Anwendung zur Verwaltung von Fotos

Das folgende Codebeispiel zeigt, wie eine Serverless-Anwendung erstellt wird, mit der Benutzer Fotos mithilfe von Labels erstellen können.

SDK für JavaScript (v3)

Zeigt, wie eine Anwendung zur Verwaltung von Fotobeständen entwickelt wird, die mithilfe von Amazon Rekognition Labels in Bildern erkennt und sie für einen späteren Abruf speichert.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

Einen tiefen Einblick in den Ursprung dieses Beispiels finden Sie im Beitrag in der [AWS - Community](#).

In diesem Beispiel verwendete Dienste

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Erstellen einer Anwendung zum Analysieren von Kundenfeedback

Das folgende Codebeispiel zeigt, wie Sie eine Anwendung erstellen, die Kundenkommentarkarten analysiert, sie aus der ursprünglichen Sprache übersetzt, die Stimmung ermittelt und auf der Grundlage des übersetzten Texts eine Audiodatei generiert.

SDK für JavaScript (v3)

Diese Beispielanwendung analysiert und speichert Kundenfeedback-Karten. Sie ist auf die Anforderungen eines fiktiven Hotels in New York City zugeschnitten. Das Hotel erhält Feedback von Gästen in Form von physischen Kommentarkarten in verschiedenen Sprachen. Dieses Feedback wird über einen Webclient in die App hochgeladen. Nachdem ein Bild einer Kommentarkarte hochgeladen wurde, werden folgende Schritte ausgeführt:

- Der Text wird mithilfe von Amazon Textract aus dem Bild extrahiert.
- Amazon Comprehend ermittelt die Stimmung und die Sprache des extrahierten Textes.
- Der extrahierte Text wird mithilfe von Amazon Translate ins Englische übersetzt.
- Amazon Polly generiert auf der Grundlage des extrahierten Texts eine Audiodatei.

Die vollständige App kann mithilfe des AWS CDK bereitgestellt werden. Quellcode und Anweisungen zur Bereitstellung finden Sie im Projekt unter [GitHub](#). Die folgenden Auszüge zeigen, wie der innerhalb von Lambda-Funktionen verwendet AWS SDK für JavaScript wird.

```
import {  
  ComprehendClient,  
  DetectDominantLanguageCommand,
```

```

    DetectSentimentCommand,
  } from "@aws-sdk/client-comprehend";

/**
 * Determine the language and sentiment of the extracted text.
 *
 * @param {{ source_text: string }} extractTextOutput
 */
export const handler = async (extractTextOutput) => {
  const comprehendClient = new ComprehendClient({});

  const detectDominantLanguageCommand = new DetectDominantLanguageCommand({
    Text: extractTextOutput.source_text,
  });

  // The source language is required for sentiment analysis and
  // translation in the next step.
  const { Languages } = await comprehendClient.send(
    detectDominantLanguageCommand,
  );

  const languageCode = Languages[0].LanguageCode;

  const detectSentimentCommand = new DetectSentimentCommand({
    Text: extractTextOutput.source_text,
    LanguageCode: languageCode,
  });

  const { Sentiment } = await comprehendClient.send(detectSentimentCommand);

  return {
    sentiment: Sentiment,
    language_code: languageCode,
  };
};

```

```

import {
  DetectDocumentTextCommand,
  TextractClient,
} from "@aws-sdk/client-textract";

/**
 * Fetch the S3 object from the event and analyze it using Amazon Textract.

```

```

*
* @param {import("@types/aws-lambda").EventBridgeEvent<"Object Created">}
eventBridgeS3Event
*/
export const handler = async (eventBridgeS3Event) => {
  const textractClient = new TextractClient();

  const detectDocumentTextCommand = new DetectDocumentTextCommand({
    Document: {
      S3Object: {
        Bucket: eventBridgeS3Event.bucket,
        Name: eventBridgeS3Event.object,
      },
    },
  });

  // Textract returns a list of blocks. A block can be a line, a page, word, etc.
  // Each block also contains geometry of the detected text.
  // For more information on the Block type, see https://docs.aws.amazon.com/
  textract/latest/dg/API_Block.html.
  const { Blocks } = await textractClient.send(detectDocumentTextCommand);

  // For the purpose of this example, we are only interested in words.
  const extractedWords = Blocks.filter((b) => b.BlockType === "WORD").map(
    (b) => b.Text,
  );

  return extractedWords.join(" ");
};

```

```

import { PollyClient, SynthesizeSpeechCommand } from "@aws-sdk/client-polly";
import { S3Client } from "@aws-sdk/client-s3";
import { Upload } from "@aws-sdk/lib-storage";

/**
 * Synthesize an audio file from text.
 *
 * @param {{ bucket: string, translated_text: string, object: string }}
sourceDestinationConfig
*/
export const handler = async (sourceDestinationConfig) => {
  const pollyClient = new PollyClient({});

```

```

const synthesizeSpeechCommand = new SynthesizeSpeechCommand({
  Engine: "neural",
  Text: sourceDestinationConfig.translated_text,
  VoiceId: "Ruth",
  OutputFormat: "mp3",
});

const { AudioStream } = await pollyClient.send(synthesizeSpeechCommand);

const audioKey = `${sourceDestinationConfig.object}.mp3`;

// Store the audio file in S3.
const s3Client = new S3Client();
const upload = new Upload({
  client: s3Client,
  params: {
    Bucket: sourceDestinationConfig.bucket,
    Key: audioKey,
    Body: AudioStream,
    ContentType: "audio/mp3",
  },
});

await upload.done();
return audioKey;
};

```

```

import {
  TranslateClient,
  TranslateTextCommand,
} from "@aws-sdk/client-translate";

/**
 * Translate the extracted text to English.
 *
 * @param {{ extracted_text: string, source_language_code: string }}
  textAndSourceLanguage
 */
export const handler = async (textAndSourceLanguage) => {
  const translateClient = new TranslateClient({});

  const translateCommand = new TranslateTextCommand({
    SourceLanguageCode: textAndSourceLanguage.source_language_code,

```

```
    TargetLanguageCode: "en",  
    Text: textAndSourceLanguage.extracted_text,  
  });  
  
  const { TranslatedText } = await translateClient.send(translateCommand);  
  
  return { translated_text: TranslatedText };  
};
```

In diesem Beispiel verwendete Dienste

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

Aufrufen einer Lambda-Funktion von einem Browser aus

Das folgende Codebeispiel zeigt, wie eine AWS Lambda Funktion von einem Browser aus aufgerufen wird.

SDK für JavaScript (v3)

Sie können eine browserbasierte Anwendung erstellen, die eine AWS Lambda Funktion verwendet, um eine Amazon DynamoDB-Tabelle mit Benutzerauswahlen zu aktualisieren. Diese App verwendet v3. AWS SDK für JavaScript

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

In diesem Beispiel verwendete Dienste

- DynamoDB
- Lambda

Verwenden von API Gateway zum Aufrufen einer Lambda-Funktion

Das folgende Codebeispiel zeigt, wie eine von Amazon API Gateway aufgerufene AWS Lambda Funktion erstellt wird.

SDK für JavaScript (v3)

Zeigt, wie eine AWS Lambda Funktion mithilfe der JavaScript Lambda-Laufzeit-API erstellt wird. In diesem Beispiel werden verschiedene AWS Dienste aufgerufen, um einen bestimmten Anwendungsfall auszuführen. Dieses Beispiel zeigt, wie man eine Lambda-Funktion erstellt, die von Amazon API Gateway aufgerufen wird und eine Amazon-DynamoDB-Tabelle nach Arbeitsjubiläen durchsucht und Amazon Simple Notification Service (Amazon SNS) verwendet, um eine Textnachricht an Ihre Mitarbeiter zu senden, die ihnen zu ihrem einjährigen Jubiläum gratuliert.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

Dieses Beispiel ist auch verfügbar im [AWS SDK für JavaScript Entwicklerhandbuch für v3](#).

In diesem Beispiel verwendete Dienste

- API Gateway
- DynamoDB
- Lambda
- Amazon SNS

Verwendung geplanter Ereignisse zum Aufrufen einer Lambda-Funktion

Das folgende Codebeispiel zeigt, wie eine AWS Lambda Funktion erstellt wird, die durch ein von Amazon EventBridge geplantes Ereignis aufgerufen wird.

SDK für JavaScript (v3)

Zeigt, wie ein von Amazon EventBridge geplantes Ereignis erstellt wird, das eine AWS Lambda Funktion aufruft. Konfigurieren Sie so EventBridge, dass ein Cron-Ausdruck verwendet wird, um zu planen, wann die Lambda-Funktion aufgerufen wird. In diesem Beispiel erstellen Sie eine Lambda-Funktion mithilfe der JavaScript Lambda-Laufzeit-API. In diesem Beispiel werden verschiedene AWS Dienste aufgerufen, um einen bestimmten Anwendungsfall auszuführen. Dieses Beispiel zeigt, wie man eine App erstellt, die eine mobile Textnachricht an Ihre Mitarbeiter sendet, um ihnen zum einjährigen Jubiläum zu gratulieren.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

Dieses Beispiel ist auch verfügbar im [AWS SDK für JavaScript Entwicklerhandbuch für v3](#).

In diesem Beispiel verwendete Dienste

- CloudWatch Logs
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

Serverless-Beispiele

Herstellen einer Verbindung mit einer Amazon-RDS-Datenbank in einer Lambda-Funktion

Die folgenden Codebeispiele veranschaulichen, wie eine Lambda-Funktion implementiert wird, die eine Verbindung zu einer RDS-Datenbank herstellt. Die Funktion stellt eine einfache Datenbankanfrage und gibt das Ergebnis zurück.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Herstellen einer Verbindung zu einer Amazon RDS-Datenbank in einer Lambda-Funktion mithilfe von JavaScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
/*
Node.js code here.
*/
// ES6+ example
import { Signer } from "@aws-sdk/rds-signer";
import mysql from 'mysql2/promise';

async function createAuthToken() {
  // Define connection authentication parameters
```

```
const dbinfo = {

  hostname: process.env.ProxyHostName,
  port: process.env.Port,
  username: process.env.DBUserName,
  region: process.env.AWS_REGION,

}

// Create RDS Signer object
const signer = new Signer(dbinfo);

// Request authorization token from RDS, specifying the username
const token = await signer.getAuthToken();
return token;
}

async function dbOps() {

  // Obtain auth token
  const token = await createAuthToken();
  // Define connection configuration
  let connectionConfig = {
    host: process.env.ProxyHostName,
    user: process.env.DBUserName,
    password: token,
    database: process.env.DBName,
    ssl: 'Amazon RDS'
  }
  // Create the connection to the DB
  const conn = await mysql.createConnection(connectionConfig);
  // Obtain the result of the query
  const [res,] = await conn.execute('select ?+? as sum', [3, 2]);
  return res;

}

export const handler = async (event) => {
  // Execute database flow
  const result = await dbOps();
  // Return result
  return {
    statusCode: 200,
    body: JSON.stringify("The selected sum is: " + result[0].sum)
  }
}
```

```
}  
};
```

Herstellen einer Verbindung zu einer Amazon RDS-Datenbank in einer Lambda-Funktion mithilfe von TypeScript.

```
import { Signer } from "@aws-sdk/rds-signer";  
import mysql from 'mysql2/promise';  
  
// RDS settings  
// Using '!' (non-null assertion operator) to tell the TypeScript compiler that the  
// DB settings are not null or undefined,  
const proxy_host_name = process.env.PROXY_HOST_NAME!  
const port = parseInt(process.env.PORT!)  
const db_name = process.env.DB_NAME!  
const db_user_name = process.env.DB_USER_NAME!  
const aws_region = process.env.AWS_REGION!  
  
async function createAuthToken(): Promise<string> {  
  
    // Create RDS Signer object  
    const signer = new Signer({  
        hostname: proxy_host_name,  
        port: port,  
        region: aws_region,  
        username: db_user_name  
    });  
  
    // Request authorization token from RDS, specifying the username  
    const token = await signer.getAuthToken();  
    return token;  
}  
  
async function dbOps(): Promise<mysql.QueryResult | undefined> {  
    try {  
        // Obtain auth token  
        const token = await createAuthToken();  
        const conn = await mysql.createConnection({  
            host: proxy_host_name,  
            user: db_user_name,
```

```

        password: token,
        database: db_name,
        ssl: 'Amazon RDS' // Ensure you have the CA bundle for SSL connection
    });
    const [rows, fields] = await conn.execute('SELECT ? + ? AS sum', [3, 2]);
    console.log('result:', rows);
    return rows;
}
catch (err) {
    console.log(err);
}
}

export const lambdaHandler = async (event: any): Promise<{ statusCode: number; body:
string }> => {
    // Execute database flow
    const result = await dbOps();

    // Return error if result is undefined
    if (result == undefined)
        return {
            statusCode: 500,
            body: JSON.stringify(`Error with connection to DB host`)
        }

    // Return result
    return {
        statusCode: 200,
        body: JSON.stringify(`The selected sum is: ${result[0].sum}`)
    };
};
};

```

Aufrufen einer Lambda-Funktion über einen Kinesis-Auslöser

Das folgende Codebeispiel veranschaulicht, wie eine Lambda-Funktion implementiert wird, die ein durch den Empfang von Datensätzen aus einem Kinesis-Stream ausgelöstes Ereignis empfängt. Die Funktion ruft die Kinesis-Nutzlast ab, dekodiert von Base64 und protokolliert den Datensatzinhalt.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Ein Kinesis-Ereignis mit Lambda verwenden. JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    try {
      console.log(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      console.log(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      console.error(`An error occurred ${err}`);
      throw err;
    }
  }
  console.log(`Successfully processed ${event.Records.length} records.`);
};

async function getRecordDataAsync(payload) {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

Ein Kinesis-Ereignis mit Lambda verwenden. TypeScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  KinesisStreamEvent,
  Context,
  KinesisStreamHandler,
```

```

    KinesisStreamRecordPayload,
  } from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";

const logger = new Logger({
  logLevel: "INFO",
  serviceName: "kinesis-stream-handler-sample",
});

export const functionHandler: KinesisStreamHandler = async (
  event: KinesisStreamEvent,
  context: Context
): Promise<void> => {
  for (const record of event.Records) {
    try {
      logger.info(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      logger.info(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      logger.error(`An error occurred ${err}`);
      throw err;
    }
    logger.info(`Successfully processed ${event.Records.length} records.`);
  }
};

async function getRecordDataAsync(
  payload: KinesisStreamRecordPayload
): Promise<string> {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}

```

Aufrufen einer Lambda-Funktion über einen DynamoDB-Auslöser

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein durch den Empfang von Datensätzen aus einem DynamoDB-Stream ausgelöstes Ereignis empfängt. Die Funktion ruft die DynamoDB-Nutzdaten ab und protokolliert den Inhalt des Datensatzes.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Ein DynamoDB-Ereignis mit Lambda verwenden. JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  console.log(JSON.stringify(event, null, 2));
  event.Records.forEach(record => {
    logDynamoDBRecord(record);
  });
};

const logDynamoDBRecord = (record) => {
  console.log(record.eventID);
  console.log(record.eventName);
  console.log(`DynamoDB Record: ${JSON.stringify(record.dynamodb)}`);
};
```

Ein DynamoDB-Ereignis mit Lambda verwenden. TypeScript

```
export const handler = async (event, context) => {
  console.log(JSON.stringify(event, null, 2));
  event.Records.forEach(record => {
    logDynamoDBRecord(record);
  });
}

const logDynamoDBRecord = (record) => {
  console.log(record.eventID);
  console.log(record.eventName);
  console.log(`DynamoDB Record: ${JSON.stringify(record.dynamodb)}`);
};
```


Aufrufen einer Lambda-Funktion über einen Amazon DocumentDB-Auslöser

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein durch den Empfang von Datensätzen aus einem DocumentDB-Änderungsstream ausgelöstes Ereignis empfängt. Die Funktion ruft die DocumentDB-Nutzdaten ab und protokolliert den Inhalt des Datensatzes.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Verwenden eines Amazon DocumentDB DocumentDB-Ereignisses mit Lambda. JavaScript

```
console.log('Loading function');
exports.handler = async (event, context) => {
  event.events.forEach(record => {
    logDocumentDBEvent(record);
  });
  return 'OK';
};

const logDocumentDBEvent = (record) => {
  console.log('Operation type: ' + record.event.operationType);
  console.log('db: ' + record.event.ns.db);
  console.log('collection: ' + record.event.ns.coll);
  console.log('Full document:', JSON.stringify(record.event.fullDocument, null,
    2));
};
```

Ein Amazon DocumentDB DocumentDB-Ereignis mit Lambda verwenden TypeScript

```
import { DocumentDBEventRecord, DocumentDBEventSubscriptionContext } from 'aws-
lambda';

console.log('Loading function');
```

```
export const handler = async (
  event: DocumentDBEventSubscriptionContext,
  context: any
): Promise<string> => {
  event.events.forEach((record: DocumentDBEventRecord) => {
    logDocumentDBEvent(record);
  });
  return 'OK';
};

const logDocumentDBEvent = (record: DocumentDBEventRecord): void => {
  console.log('Operation type: ' + record.event.operationType);
  console.log('db: ' + record.event.ns.db);
  console.log('collection: ' + record.event.ns.coll);
  console.log('Full document:', JSON.stringify(record.event.fullDocument, null, 2));
};
```

Aufrufen einer Lambda-Funktion über einen Amazon-MSK-Auslöser

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein Ereignis empfängt, das durch den Empfang von Datensätzen von einem Amazon MSK-Cluster ausgelöst wird. Die Funktion ruft die MSK-Nutzdaten ab und protokolliert den Inhalt des Datensatzes.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Ein Amazon MSK-Ereignis mit Lambda verwenden. JavaScript

```
exports.handler = async (event) => {
  // Iterate through keys
  for (let key in event.records) {
    console.log('Key: ', key)
    // Iterate through records
    event.records[key].map((record) => {
```

```
        console.log('Record: ', record)
        // Decode base64
        const msg = Buffer.from(record.value, 'base64').toString()
        console.log('Message:', msg)
    })
}
```

Ein Amazon MSK-Ereignis mit Lambda verwenden. TypeScript

```
import { MSKEvent, Context } from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";

const logger = new Logger({
  logLevel: "INFO",
  serviceName: "msk-handler-sample",
});

export const handler = async (
  event: MSKEvent,
  context: Context
): Promise<void> => {
  for (const [topic, topicRecords] of Object.entries(event.records)) {
    logger.info(`Processing key: ${topic}`);

    // Process each record in the partition
    for (const record of topicRecords) {
      try {
        // Decode the message value from base64
        const decodedMessage = Buffer.from(record.value, 'base64').toString();

        logger.info({
          message: decodedMessage
        });
      }
      catch (error) {
        logger.error('Error processing event', { error });
        throw error;
      }
    }
  }
}
```

Aufrufen einer Lambda-Funktion über einen Amazon-S3-Auslöser

Im folgenden Codebeispiel wird die Implementierung einer Lambda-Funktion gezeigt, die ein Ereignis empfängt, das durch Hochladen eines Objekts in einen S3-Bucket ausgelöst wird. Die Funktion ruft den Namen des S3-Buckets sowie den Objektschlüssel aus dem Ereignisparameter ab und ruft die Amazon-S3-API auf, um den Inhaltstyp des Objekts abzurufen und zu protokollieren.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Konsumieren eines S3-Ereignisses mit Lambda unter Verwendung JavaScript.

```
import { S3Client, HeadObjectCommand } from "@aws-sdk/client-s3";

const client = new S3Client();

export const handler = async (event, context) => {

  // Get the object from the event and show its content type
  const bucket = event.Records[0].s3.bucket.name;
  const key = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, ' '));

  try {
    const { ContentType } = await client.send(new HeadObjectCommand({
      Bucket: bucket,
      Key: key,
    }));

    console.log('CONTENT TYPE:', ContentType);
    return ContentType;

  } catch (err) {
    console.log(err);
  }
}
```

```
    const message = `Error getting object ${key} from bucket ${bucket}. Make
    sure they exist and your bucket is in the same region as this function.`;
    console.log(message);
    throw new Error(message);
  }
};
```

Konsumieren eines S3-Ereignisses mit Lambda unter Verwendung TypeScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { S3Event } from 'aws-lambda';
import { S3Client, HeadObjectCommand } from '@aws-sdk/client-s3';

const s3 = new S3Client({ region: process.env.AWS_REGION });

export const handler = async (event: S3Event): Promise<string | undefined> => {
  // Get the object from the event and show its content type
  const bucket = event.Records[0].s3.bucket.name;
  const key = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, '
  '));
  const params = {
    Bucket: bucket,
    Key: key,
  };
  try {
    const { ContentType } = await s3.send(new HeadObjectCommand(params));
    console.log('CONTENT TYPE:', ContentType);
    return ContentType;
  } catch (err) {
    console.log(err);
    const message = `Error getting object ${key} from bucket ${bucket}. Make sure
    they exist and your bucket is in the same region as this function.`;
    console.log(message);
    throw new Error(message);
  }
};
```

Eine Lambda-Funktion über einen Amazon-SNS-Trigger aufrufen

Im folgenden Codebeispiel wird die Implementierung einer Lambda-Funktion veranschaulicht, die ein Ereignis empfängt, das durch das Empfangen von Nachrichten aus einem SNS-Thema ausgelöst wird. Die Funktion ruft die Nachrichten aus dem Ereignisparameter ab und protokolliert den Inhalt jeder Nachricht.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Konsumieren eines SNS-Ereignisses mit Lambda unter Verwendung. JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    await processMessageAsync(record);
  }
  console.info("done");
};

async function processMessageAsync(record) {
  try {
    const message = JSON.stringify(record.Sns.Message);
    console.log(`Processed message ${message}`);
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

Konsumieren eines SNS-Ereignisses mit Lambda unter Verwendung. TypeScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
```

```
// SPDX-License-Identifier: Apache-2.0
import { SNSEvent, Context, SNSHandler, SNSEventRecord } from "aws-lambda";

export const functionHandler: SNSHandler = async (
  event: SNSEvent,
  context: Context
): Promise<void> => {
  for (const record of event.Records) {
    await processMessageAsync(record);
  }
  console.info("done");
};

async function processMessageAsync(record: SNSEventRecord): Promise<any> {
  try {
    const message: string = JSON.stringify(record.Sns.Message);
    console.log(`Processed message ${message}`);
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

Aufrufen einer Lambda-Funktion über einen Amazon-SQS-Auslöser

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein Ereignis empfängt, das durch den Empfang von Nachrichten aus einer SQS-Warteschlange ausgelöst wird. Die Funktion ruft die Nachrichten aus dem Ereignisparameter ab und protokolliert den Inhalt jeder Nachricht.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Konsumieren eines SQS-Ereignisses mit Lambda unter Verwendung. JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const message of event.Records) {
    await processMessageAsync(message);
  }
  console.info("done");
};

async function processMessageAsync(message) {
  try {
    console.log(`Processed message ${message.body}`);
    // TODO: Do interesting work based on the new message
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

Konsumieren eines SQS-Ereignisses mit Lambda unter Verwendung. TypeScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { SQSEvent, Context, SQSHandler, SQSRecord } from "aws-lambda";

export const functionHandler: SQSHandler = async (
  event: SQSEvent,
  context: Context
): Promise<void> => {
  for (const message of event.Records) {
    await processMessageAsync(message);
  }
  console.info("done");
};

async function processMessageAsync(message: SQSRecord): Promise<any> {
  try {
    console.log(`Processed message ${message.body}`);
    // TODO: Do interesting work based on the new message
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
```



```
    console.error("An error occurred");
    throw err;
  }
}
```

Melden von Batch-Elementfehlern für Lambda-Funktionen mit einem Kinesis-Auslöser

Das folgende Codebeispiel zeigt, wie eine teilweise Batch-Antwort für Lambda-Funktionen implementiert wird, die Ereignisse von einem Kinesis-Stream empfangen. Die Funktion meldet die Batch-Elementfehler in der Antwort und signalisiert Lambda, diese Nachrichten später erneut zu versuchen.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Melden von Fehlern bei Kinesis-Batchelementen mit Lambda unter Verwendung von Javascript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    try {
      console.log(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      console.log(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    } catch (err) {
      console.error(`An error occurred ${err}`);
      /* Since we are working with streams, we can return the failed item
      immediately.
      Lambda will immediately begin to retry processing from this failed item
      onwards. */
      return {
        batchItemFailures: [{ itemIdentifier: record.kinesis.sequenceNumber }],
      };
    }
  }
}
```

```

    }
  }
  console.log(`Successfully processed ${event.Records.length} records.`);
  return { batchItemFailures: [] };
};

async function getRecordDataAsync(payload) {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}

```

Melden von Fehlern Kinesis Kinesis-Batch-Elementen mit Lambda unter Verwendung von TypeScript

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import {
  KinesisStreamEvent,
  Context,
  KinesisStreamHandler,
  KinesisStreamRecordPayload,
  KinesisStreamBatchResponse,
} from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";

const logger = new Logger({
  logLevel: "INFO",
  serviceName: "kinesis-stream-handler-sample",
});

export const functionHandler: KinesisStreamHandler = async (
  event: KinesisStreamEvent,
  context: Context
): Promise<KinesisStreamBatchResponse> => {
  for (const record of event.Records) {
    try {
      logger.info(`Processed Kinesis Event - EventID: ${record.eventID}`);
      const recordData = await getRecordDataAsync(record.kinesis);
      logger.info(`Record Data: ${recordData}`);
      // TODO: Do interesting work based on the new data
    }
  }
}

```

```
    } catch (err) {
      logger.error(`An error occurred ${err}`);
      /* Since we are working with streams, we can return the failed item
      immediately.
      Lambda will immediately begin to retry processing from this failed item
      onwards. */
      return {
        batchItemFailures: [{ itemIdentifier: record.kinesis.sequenceNumber }],
      };
    }
  }
  logger.info(`Successfully processed ${event.Records.length} records.`);
  return { batchItemFailures: [] };
};

async function getRecordDataAsync(
  payload: KinesisStreamRecordPayload
): Promise<string> {
  var data = Buffer.from(payload.data, "base64").toString("utf-8");
  await Promise.resolve(1); //Placeholder for actual async work
  return data;
}
```

Melden von Batch-Elementfehlern für Lambda-Funktionen mit einem DynamoDB-Auslöser

Das folgende Codebeispiel zeigt, wie eine teilweise Batch-Antwort für Lambda-Funktionen implementiert wird, die Ereignisse von einem DynamoDB-Stream empfangen. Die Funktion meldet die Batch-Elementfehler in der Antwort und signalisiert Lambda, diese Nachrichten später erneut zu versuchen.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Melden von DynamoDB-Batchelementfehlern mit Lambda unter Verwendung von. JavaScript

```
export const handler = async (event) => {
  const records = event.Records;
  let curRecordSequenceNumber = "";

  for (const record of records) {
    try {
      // Process your record
      curRecordSequenceNumber = record.dynamodb.SequenceNumber;
    } catch (e) {
      // Return failed record's sequence number
      return { batchItemFailures: [{ itemIdentifier: curRecordSequenceNumber }] };
    }
  }

  return { batchItemFailures: [] };
};
```

Melden von DynamoDB-Batchelementfehlern mit Lambda unter Verwendung von TypeScript

```
import {
  DynamoDBBatchResponse,
  DynamoDBBatchItemFailure,
  DynamoDBStreamEvent,
} from "aws-lambda";

export const handler = async (
  event: DynamoDBStreamEvent
): Promise<DynamoDBBatchResponse> => {
  const batchItemFailures: DynamoDBBatchItemFailure[] = [];
  let curRecordSequenceNumber;

  for (const record of event.Records) {
    curRecordSequenceNumber = record.dynamodb?.SequenceNumber;

    if (curRecordSequenceNumber) {
      batchItemFailures.push({
        itemIdentifier: curRecordSequenceNumber,
      });
    }
  }

  return { batchItemFailures: batchItemFailures };
};
```

```
};
```

Melden von Batch-Elementfehlern für Lambda-Funktionen mit einem Amazon-SQS-Auslöser

Das folgende Codebeispiel zeigt, wie eine teilweise Batch-Antwort für Lambda-Funktionen implementiert wird, die Ereignisse aus einer SQS-Warteschlange empfangen. Die Funktion meldet die Batch-Elementfehler in der Antwort und signalisiert Lambda, diese Nachrichten später erneut zu versuchen.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Melden von SQS-Batch-Elementfehlern mit Lambda unter Verwendung von JavaScript

```
// Node.js 20.x Lambda runtime, AWS SDK for Javascript V3
export const handler = async (event, context) => {
  const batchItemFailures = [];
  for (const record of event.Records) {
    try {
      await processMessageAsync(record, context);
    } catch (error) {
      batchItemFailures.push({ itemIdentifier: record.messageId });
    }
  }
  return { batchItemFailures };
};

async function processMessageAsync(record, context) {
  if (record.body && record.body.includes("error")) {
    throw new Error("There is an error in the SQS Message.");
  }
  console.log(`Processed message: ${record.body}`);
}
```

Melden von SQS-Batch-Elementfehlern mit Lambda unter Verwendung von TypeScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { SQSEvent, SQSBatchResponse, Context, SQSBatchItemFailure, SQSRecord } from
  'aws-lambda';

export const handler = async (event: SQSEvent, context: Context):
  Promise<SQSBatchResponse> => {
  const batchItemFailures: SQSBatchItemFailure[] = [];

  for (const record of event.Records) {
    try {
      await processMessageAsync(record);
    } catch (error) {
      batchItemFailures.push({ itemIdentifier: record.messageId });
    }
  }

  return {batchItemFailures: batchItemFailures};
};

async function processMessageAsync(record: SQSRecord): Promise<void> {
  if (record.body && record.body.includes("error")) {
    throw new Error('There is an error in the SQS Message.');
```

Amazon Lex Lex-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK für JavaScript (v3) mit Amazon Lex Aktionen ausführen und allgemeine Szenarien implementieren.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb eines Service aufrufen oder mit anderen AWS-Services kombinieren.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kodex finden.

Themen

- [Szenarien](#)

Szenarien

Erstellen eines Amazon-Lex-Chatbots

Das folgende Codebeispiele zeigt, wie Sie einen Chatbot erstellen können, um die Besucher Ihrer Website einzubinden.

SDK für JavaScript (v3)

Zeigt, wie Sie die Amazon-Lex-API verwenden können, um einen Chatbot innerhalb einer Webanwendung zu erstellen, um die Besucher Ihrer Website einzubinden.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel „[Einen Amazon Lex-Chatbot erstellen](#)“ im AWS SDK für JavaScript Entwicklerhandbuch.

In diesem Beispiel verwendete Dienste

- Amazon Comprehend
- Amazon Lex
- Amazon Translate

Beispiele für Amazon-Standorte mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie AWS SDK für JavaScript (v3) mit Amazon Location verwenden.

Bei Grundlagen handelt es sich um Codebeispiele, die Ihnen zeigen, wie Sie die wesentlichen Vorgänge innerhalb eines Services ausführen.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kodex finden.

Themen

- [Erste Schritte](#)

- [Grundlagen](#)
- [Aktionen](#)

Erste Schritte

Hello Amazon Location

Das folgende Codebeispiel zeigt, wie Sie mit Amazon Location Service beginnen können.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { fileURLToPath } from "node:url";
import {
  LocationClient,
  ListGeofenceCollectionsCommand,
} from "@aws-sdk/client-location";

/**
 * Lists geofences from a specified geofence collection asynchronously.
 */
export const main = async () => {
  const region = "eu-west-1";
  const locationClient = new LocationClient({ region: region });
  const listGeofenceCollParams = {
    MaxResults: 100,
  };
  try {
    const command = new ListGeofenceCollectionsCommand(listGeofenceCollParams);
    const response = await locationClient.send(command);
    const geofenceEntries = response.Entries;
    if (geofenceEntries.length === 0) {
      console.log("No Geofences were found in the collection.");
    } else {
      for (const geofenceEntry of geofenceEntries) {
        console.log(`Geofence ID: ${geofenceEntry.CollectionName}`);
      }
    }
  }
};
```



```
    }  
  }  
} catch (error) {  
  console.error(  
    `A validation error occurred while creating geofence: ${error} \n Exiting  
program.`,  
  );  
  return;  
}  
};
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für JavaScript - API-Referenz.
 - [ListGeofenceCollections](#)
 - [ListGeofences](#)

Grundlagen

Kennenlernen der Grundlagen

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Erstellen Sie eine Amazon-Location-Karte.
- Erstellen Sie einen API-Schlüssel für Amazon Location.
- Zeigen Sie die Karten-URL an.
- Erstellen Sie eine Geofence-Sammlung.
- Speichern einer Geofence-Geometrie
- Erstellen einer Tracker-Ressource
- Aktualisieren der Position eines Geräts
- Rufen Sie die neueste Positionsaktualisierung für ein bestimmtes Gerät ab.
- Erstellen Sie einen Routenrechner.
- Ermitteln der Entfernung zwischen Seattle und Vancouver
- Verwenden Sie Amazon Location auf höherer Ebene APIs.
- Löschen Sie die Amazon-Location-Komponenten.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
/*
Before running this JavaScript code example, set up your development environment,
including your credentials.
This demo illustrates how to use the AWS SDK for JavaScript (v3) to work with Amazon
Location Service.

For more information, see the following documentation topic:

https://docs.aws.amazon.com/sdk-for-javascript/v3/developer-guide/getting-
started.html
*/

import {
  Scenario,
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";

import {
  CreateMapCommand,
  CreateGeofenceCollectionCommand,
  PutGeofenceCommand,
  CreateTrackerCommand,
  BatchUpdateDevicePositionCommand,
  GetDevicePositionCommand,
  CreateRouteCalculatorCommand,
  CalculateRouteCommand,
  LocationClient,
  ConflictException,
  ResourceNotFoundException,
  DeleteGeofenceCollectionCommand,
  DeleteRouteCalculatorCommand,
```

```
    DeleteTrackerCommand,
    DeleteMapCommand,
} from "@aws-sdk/client-location";

import {
    GeoPlacesClient,
    ReverseGeocodeCommand,
    SearchNearbyCommand,
    SearchTextCommand,
    GetPlaceCommand,
    ValidationException,
} from "@aws-sdk/client-geo-places";

import { parseArgs } from "node:util";
import { fileURLToPath } from "node:url";

/*The inputs for this example can be edited in the ./input.json.*/
import data from "./inputs.json" with { type: "json" };

/**
 * Used repeatedly to have the user press enter.
 * @type {ScenarioInput}
 */
/* v8 ignore next 3 */
const pressEnter = new ScenarioInput("continue", "Press Enter to continue", {
    type: "confirm",
    verbose: "false",
});

const pressEnterConfirm = new ScenarioInput(
    "confirm",
    "Press Enter to continue",
    {
        type: "confirm",
        verbose: "false",
    },
);

const region = "eu-west-1";

const locationClient = new LocationClient({ region: region });

const greet = new ScenarioOutput(
    "greet",
```

```

    "Welcome to the Amazon Location Use demo! \n" +
    "AWS Location Service is a fully managed service offered by Amazon Web Services (AWS) that " +
    "provides location-based services for developers. This service simplifies " +
    "the integration of location-based features into applications, making it " +
    "Maps: The service provides access to high-quality maps, satellite imagery, " +
    "and geospatial data from various providers, allowing developers to " +
    "easily embed maps into their applications:\n" +
    "Tracking: The Location Service enables real-time tracking of mobile devices, "
+
    "assets, or other entities, allowing developers to build applications " +
    "that can monitor the location of people, vehicles, or other objects.\n" +
    "Geocoding: The service provides the ability to convert addresses or " +
    "location names into geographic coordinates (latitude and longitude), " +
    "and vice versa, enabling developers to integrate location-based search " +
    "and routing functionality into their applications. " +
    "Please define values ./inputs.json for each user-defined variable used in this app. Otherwise the default is used:\n" +
    "- mapName: The name of the map to be create (default is 'AWSMap').\n" +
    "- keyName: The name of the API key to create (default is 'AWSApiKey')\n" +
    "- collectionName: The name of the geofence collection (default is 'AWSLocationCollection')\n" +
    "- geoId: The geographic identifier used for the geofence or map (default is 'geoId')\n" +
    "- trackerName: The name of the tracker (default is 'geoTracker')\n" +
    "- calculatorName: The name of the route calculator (default is 'AWSRouteCalc')\n" +
    "- deviceId: The ID of the device (default is 'iPhone-112356')",

    { header: true },
);

const displayCreateAMap = new ScenarioOutput(
    "displayCreateAMap",
    "1. Create a map\n" +
    "An AWS Location map can enhance the user experience of your " +
    "application by providing accurate and personalized location-based " +
    "features. For example, you could use the geocoding capabilities to " +
    "allow users to search for and locate businesses, landmarks, or " +
    "other points of interest within a specific region.",
);

const sdkCreateAMap = new ScenarioAction(
    "sdkCreateAMap",
    async (** @type {State} */ state) => {

```

```

const createMapParams = {
  MapName: `${data.inputs.mapName}`,
  Configuration: { style: "VectorEsriNavigation" },
};
try {
  const command = new CreateMapCommand(createMapParams);
  const response = await locationClient.send(command);
  state.MapName = response.MapName;
  console.log("Map created. Map ARN is: ", state.MapName);
} catch (error) {
  console.error("Error creating map: ", error);
  throw error;
}
},
);

const displayMapUrl = new ScenarioOutput(
  "displayMapUrl",
  "2. Display Map URL\n" +
    "When you embed a map in a web app or website, the API key is " +
    "included in the map tile URL to authenticate requests. You can " +
    "restrict API keys to specific AWS Location operations (e.g., only " +
    "maps, not geocoding). API keys can expire, ensuring temporary " +
    "access control.\n" +
    "In order to get the MAP URL you need to create and get the API Key value. " +
    "You can create and get the key value using the AWS Management Console under " +
    "Location Services. These operations cannot be completed using the " +
    "AWS SDK. For more information about getting the key value, see " +
    "the AWS Location Documentation.",
);

const sdkDisplayMapUrl = new ScenarioAction(
  "sdkDisplayMapUrl",
  async (** @type {State} */ state) => {
    const mapURL = `https://maps.geo.aws.amazon.com/maps/v0/maps/${state.MapName}/` +
      `tiles/{z}/{x}/{y}?key=API_KEY_VALUE`;
    state.mapURL = mapURL;
    console.log(
      `Replace \'API_KEY_VALUE\' in the following URL with the value for the API key` +
      `you create and get from the AWS Management Console under Location Services. This is` +
      `then the Map URL you can embed this URL in your Web app:\n` +
      `${state.mapURL}`,
    );
  },
);

```

```

);
const displayCreateGeoFenceColl = new ScenarioOutput(
  "displayCreateGeoFenceColl",
  "3. Create a geofence collection, which manages and stores geofences.",
);

const sdkCreateGeoFenceColl = new ScenarioAction(
  "sdkCreateGeoFenceColl",
  async (** @type {State} */ state) => {
    // Creates a new geofence collection.
    const geoFenceCollParams = {
      CollectionName: `${data.inputs.collectionName}`,
    };
    try {
      const command = new CreateGeofenceCollectionCommand(geoFenceCollParams);
      const response = await locationClient.send(command);
      state.CollectionName = response.CollectionName;
      console.log(
        `The geofence collection was successfully created: ${state.CollectionName}`,
      );
    } catch (caught) {
      if (caught instanceof ConflictException) {
        console.error(
          `An unexpected error occurred while creating the geofence collection:
${caught.message} \n Exiting program.`,
        );
        return;
      }
    }
  },
);

const displayStoreGeometry = new ScenarioOutput(
  "displayStoreGeometry",
  "4. Store a geofence geometry in a given geofence collection. " +
  "An AWS Location geofence is a virtual boundary that defines a geographic area "
+
  "on a map. It is a useful feature for tracking the location of " +
  "assets or monitoring the movement of objects within a specific region. " +
  "To define a geofence, you need to specify the coordinates of a " +
  "polygon that represents the area of interest. The polygon must be " +
  "defined in a counter-clockwise direction, meaning that the points of " +
  "the polygon must be listed in a counter-clockwise order. " +
  "This is a requirement for the AWS Location service to correctly " +
  "interpret the geofence and ensure that the location data is " +

```

```

    "accurately processed within the defined area.",
  );

const sdkStoreGeometry = new ScenarioAction(
  "sdkStoreGeometry",
  async (/** @type {State} */ state) => {
    const geoFenceGeoParams = {
      CollectionName: `${data.inputs.collectionName}`,
      GeofenceId: `${data.inputs.geoId}`,
      Geometry: {
        Polygon: [
          [
            [-122.3381, 47.6101],
            [-122.3281, 47.6101],
            [-122.3281, 47.6201],
            [-122.3381, 47.6201],
            [-122.3381, 47.6101],
          ],
        ],
      },
    };
    try {
      const command = new PutGeofenceCommand(geoFenceGeoParams);
      const response = await locationClient.send(command);
      state.GeoFenceId = response.GeofenceId;
      console.log("GeoFence created. GeoFence ID is: ", state.GeoFenceId);
    } catch (caught) {
      if (caught instanceof ValidationException) {
        console.error(
          `A validation error occurred while creating geofence: ${caught.message} \n
          Exiting program.`);
      }
      return;
    }
  },
);

const displayCreateTracker = new ScenarioOutput(
  "displayCreateTracker",
  "5. Create a tracker resource which lets you retrieve current and historical
  location of devices.",
);

const sdkCreateTracker = new ScenarioAction(

```

```

    "sdkCreateTracker",
    async (** @type {State} */ state) => {
        //Creates a new tracker resource in your AWS account, which you can use to track
        the location of devices.
        const createTrackerParams = {
            TrackerName: `${data.inputs.trackerName}`,
            Description: "Created using the JavaScript V3 SDK",
            PositionFiltering: "TimeBased",
        };
        try {
            const command = new CreateTrackerCommand(createTrackerParams);
            const response = await locationClient.send(command);
            state.trackerName = response.TrackerName;
            console.log("Tracker created. Tracker name is : ", state.trackerName);
        } catch (caught) {
            if (caught instanceof ResourceNotFoundException) {
                console.error(
                    `A validation error occurred while creating geofence: ${caught.message} \n
                    Exiting program.`);
            } else {
                `An unexpected error error occurred: ${caught.message} \n Exiting program.`;
            }
            return;
        }
    },
);

const displayUpdatePosition = new ScenarioOutput(
    "displayUpdatePosition",
    "6. Update the position of a device in the location tracking system." +
    "The AWS Location Service does not enforce a strict format for deviceId, but it
    must:\n " +
    "- Be a string (case-sensitive).\n" +
    "- Be 1-100 characters long.\n" +
    "- Contain only: Alphanumeric characters (A-Z, a-z, 0-9); Underscores (_);
    Hyphens (-); and be the same ID used when sending and retrieving positions.",
);

const sdkUpdatePosition = new ScenarioAction(
    "sdkUpdatePosition",
    async (** @type {State} */ state) => {
        // Updates the position of a device in the location tracking system.

        const updateDevicePosParams = {

```



```

    TrackerName: `${data.inputs.trackerName}`,
    Updates: [
      {
        DeviceId: `${data.inputs.deviceId}`,
        SampleTime: new Date(),
        Position: [-122.4194, 37.7749],
      },
    ],
  };
  try {
    const command = new BatchUpdateDevicePositionCommand(
      updateDevicePosParams,
    );
    const response = await locationClient.send(command);
    console.log(
      `Device with id ${data.inputs.deviceId} was successfully updated in the
location tracking system. `,
    );
  } catch (caught) {
    if (caught instanceof ResourceNotFoundException) {
      console.error(
        `A validation error occurred while updating the device: ${caught.message}
\n Exiting program.`,
      );
    }
  }
};

const displayRetrievePosition = new ScenarioOutput(
  "displayRetrievePosition",
  "7. Retrieve the most recent position update for a specified device.",
);

const sdkRetrievePosition = new ScenarioAction(
  "sdkRetrievePosition",
  async (/** @type {State} */ state) => {
    const devicePositionParams = {
      TrackerName: `${data.inputs.trackerName}`,
      DeviceId: `${data.inputs.deviceId}`,
    };
    try {
      const command = new GetDevicePositionCommand(devicePositionParams);
      const response = await locationClient.send(command);
      state.position = response.Position;
    }
  }
);

```

```

        console.log("Successfully fetched device position: : ", state.position);
    } catch (caught) {
        if (caught instanceof ResourceNotFoundException) {
            console.error(
                `The resource was not found: ${caught.message} \n Exiting program.`
            );
        } else {
            `An unexpected error error occurred: ${caught.message} \n Exiting program.`;
        }
        return;
    }
},
);
const displayCreateRouteCalc = new ScenarioOutput(
    "displayCreateRouteCalc",
    "8. Create a route calculator.",
);

const sdkCreateRouteCalc = new ScenarioAction(
    "sdkCreateRouteCalc",
    async (** @type {State} */ state) => {
        const routeCalcParams = {
            CalculatorName: `${data.inputs.calculatorName}`,
            DataSource: "Esri",
        };
        try {
            // Creates a new route calculator with the specified name and data source.
            const command = new CreateRouteCalculatorCommand(routeCalcParams);
            const response = await locationClient.send(command);
            state.CalculatorName = response.CalculatorName;
            console.log(
                "Route calculator created successfully. Calculator name is: ",
                state.CalculatorName,
            );
        } catch (caught) {
            if (caught instanceof ConflictException) {
                console.error(
                    `An conflict occurred: ${caught.message} \n Exiting program.`
                );
                return;
            }
        }
    },
);

```

```

const displayDetermineDist = new ScenarioOutput(
  "displayDetermineDist",
  "9. Determine the distance between Seattle and Vancouver using the route calculator.",
);

const sdkDetermineDist = new ScenarioAction(
  "sdkDetermineDist",
  async (** @type {State} */ state) => {
    // Calculates the distance between two locations asynchronously.
    const determineDist = {
      CalculatorName: `${data.inputs.calculatorName}`,
      DeparturePosition: [-122.3321, 47.6062],
      DestinationPosition: [-123.1216, 49.2827],
      TravelMode: "Car",
      DistanceUnit: "Kilometers",
    };
    try {
      const command = new CalculateRouteCommand(determineDist);
      const response = await locationClient.send(command);

      console.log(
        "Successfully calculated route. The distance in kilometers is : ",
        response.Summary.Distance,
      );
    } catch (caught) {
      if (caught instanceof ResourceNotFoundException) {
        console.error(
          `Failed to calculate route: ${caught.message} \n Exiting program.`
        );
      }
      return;
    }
  },
);

const displayUseGeoPlacesClient = new ScenarioOutput(
  "displayUseGeoPlacesClient",
  "10. Use the GeoPlacesAsyncClient to perform additional operations. " +
    "This scenario will show use of the GeoPlacesClient that enables " +
    "location search and geocoding capabilities for your applications. " +
    "We are going to use this client to perform these AWS Location tasks: \n" +
    " - Reverse Geocoding (reverseGeocode): Converts geographic coordinates into addresses.\n " +
    " - Place Search (searchText): Finds places based on search queries.\n " +

```

```

    " - Nearby Search (searchNearby): Finds places near a specific location.\n " +
    "First we will perform a Reverse Geocoding operation",
  );

const sdkUseGeoPlacesClient = new ScenarioAction(
  "sdkUseGeoPlacesClient",
  async (/** @type {State} */ state) => {
    const geoPlacesClient = new GeoPlacesClient({ region: region });

    const reverseGeoCodeParams = {
      QueryPosition: [-122.4194, 37.7749],
    };
    const searchTextParams = {
      QueryText: "coffee shop",
      BiasPosition: [-122.4194, 37.7749], //San Fransisco
    };
    const searchNearbyParams = {
      QueryPosition: [-122.4194, 37.7749],
      QueryRadius: Number("1000"),
    };
    try {
      /* Performs reverse geocoding using the AWS Geo Places API.
      Reverse geocoding is the process of converting geographic coordinates (latitude
      and longitude) to a human-readable address.
      This method uses the latitude and longitude of San Francisco as the input, and
      prints the resulting address.*/

      console.log("Use latitude 37.7749 and longitude -122.4194.");
      const command = new ReverseGeocodeCommand(reverseGeoCodeParams);
      const response = await geoPlacesClient.send(command);
      console.log(
        "Successfully calculated route. The distance in kilometers is : ",
        response.ResultItems[0].Distance,
      );
    } catch (caught) {
      if (caught instanceof ValidationException) {
        console.error(
          `An conflict occurred: ${caught.message} \n Exiting program.`
        );
        return;
      }
    }
    try {
      console.log(

```

```

        "Now we are going to perform a text search using coffee shop",
    );

    /*Searches for a place using the provided search query and prints the detailed
    information of the first result.
    @param searchTextParams the search query to be used for the place search (ex,
    coffee shop)*/

    const command = new SearchTextCommand(searchTextParams);
    const response = await geoPlacesClient.send(command);
    const placeId = response.ResultItems[0].PlaceId.toString();
    const getPlaceCommand = new GetPlaceCommand({
        PlaceId: placeId,
    });
    const getPlaceResponse = await geoPlacesClient.send(getPlaceCommand);
    console.log(
        `Detailed Place Information: \n Name and address:
    ${getPlaceResponse.Address.Label}`,
    );

    const foodTypes = getPlaceResponse.FoodTypes;
    if (foodTypes.length) {
        console.log("Food Types: ");
        for (const foodType of foodTypes) {
            console.log("- ", foodType.LocalizedName);
        }
    } else {
        console.log("No food types available.");
    }
} catch (caught) {
    if (caught instanceof ValidationException) {
        console.error(
            `An conflict occurred: ${caught.message} \n Exiting program.`
        );
        return;
    }
}
try {
    console.log("\nNow we are going to perform a nearby search.");
    const command = new SearchNearbyCommand(searchNearbyParams);
    const response = await geoPlacesClient.send(command);
    const resultItems = response.ResultItems;
    console.log("\nSuccessfully performed nearby search.");
    for (const resultItem of resultItems) {

```

```

        console.log("Name and address: ", resultItem.Address.Label);
        console.log("Distance: ", resultItem.Distance);
    }
} catch (caught) {
    if (caught instanceof ValidationException) {
        console.error(
            `An conflict occurred: ${caught.message} \n Exiting program.`
        );
        return;
    }
}
},
);

const displayDeleteResources = new ScenarioOutput(
    "displayDeleteResources",
    "11. Delete the AWS Location Services resources. " +
    "Would you like to delete the AWS Location Services resources? (y/n)",
);

const sdkDeleteResources = new ScenarioAction(
    "sdkDeleteResources",
    async (/** @type {State} */ state) => {
        const deleteGeofenceCollParams = {
            CollectionName: `${state.CollectionName}`,
        };
        const deleteRouteCalculatorParams = {
            CalculatorName: `${state.CalculatorName}`,
        };
        const deleteTrackerParams = { TrackerName: `${state.trackerName}` };
        const deleteMapParams = { MapName: `${state.MapName}` };
        try {
            const command = new DeleteMapCommand(deleteMapParams);
            const response = await locationClient.send(command);
            console.log("Map deleted.");
        } catch (error) {
            console.log("Error deleting map: ", error);
        }
        try {
            const command = new DeleteGeofenceCollectionCommand(
                deleteGeofenceCollParams,
            );
            const response = await locationClient.send(command);
            console.log("Geofence collection deleted.");
        }
    }
);

```

```
    } catch (error) {
      console.log("Error deleting geofence collection: ", error);
    }
    try {
      const command = new DeleteRouteCalculatorCommand(
        deleteRouteCalculatorParams,
      );
      const response = await locationClient.send(command);
      console.log("Route calculator deleted.");
    } catch (error) {
      console.log("Error deleting route calculator: ", error);
    }
    try {
      const command = new DeleteTrackerCommand(deleteTrackerParams);
      const response = await locationClient.send(command);
      console.log("Tracker deleted.");
    } catch (error) {
      console.log("Error deleting tracker: ", error);
    }
  },
);

const goodbye = new ScenarioOutput(
  "goodbye",
  "Thank you for checking out the Amazon Location Service Use demo. We hope you " +
    "learned something new, or got some inspiration for your own apps today!" +
    " For more Amazon Location Services examples in different programming languages, have a look at: " +
    "https://docs.aws.amazon.com/code-library/latest/ug/location_code_examples.html",
);

const myScenario = new Scenario("Location Services Scenario", [
  greet,
  pressEnter,
  displayCreateAMap,
  sdkCreateAMap,
  pressEnter,
  displayMapUrl,
  sdkDisplayMapUrl,
  pressEnter,
  displayCreateGeoFenceColl,
  sdkCreateGeoFenceColl,
  pressEnter,
```

```
displayStoreGeometry,
sdkStoreGeometry,
pressEnter,
displayCreateTracker,
sdkCreateTracker,
pressEnter,
displayUpdatePosition,
sdkUpdatePosition,
pressEnter,
displayRetrievePosition,
sdkRetrievePosition,
pressEnter,
displayCreateRouteCalc,
sdkCreateRouteCalc,
pressEnter,
displayDetermineDist,
sdkDetermineDist,
pressEnter,
displayUseGeoPlacesClient,
sdkUseGeoPlacesClient,
pressEnter,
displayDeleteResources,
pressEnterConfirm,
sdkDeleteResources,
goodbye,
]);

/** @type {{ stepHandlerOptions: StepHandlerOptions }} */
export const main = async (stepHandlerOptions) => {
  await myScenario.run(stepHandlerOptions);
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const { values } = parseArgs({
    options: {
      yes: {
        type: "boolean",
        short: "y",
      },
    },
  });
  main({ confirmAll: values.yes });
}
```


Aktionen

BatchUpdateDevicePosition

Das folgende Codebeispiel zeigt die Verwendung `BatchUpdateDevicePosition`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { fileURLToPath } from "node:url";
import {
  BatchUpdateDevicePositionCommand,
  LocationClient,
  ResourceNotFoundException,
} from "@aws-sdk/client-location";
import data from "./inputs.json" with { type: "json" };
const region = "eu-west-1";
const locationClient = new LocationClient({ region: region });
const updateDevicePosParams = {
  TrackerName: `${data.inputs.trackerName}`,
  Updates: [
    {
      DeviceId: `${data.inputs.deviceId}`,
      SampleTime: new Date(),
      Position: [-122.4194, 37.7749],
    },
  ],
};
export const main = async () => {
  try {
    const command = new BatchUpdateDevicePositionCommand(updateDevicePosParams);
    const response = await locationClient.send(command);
    //console.log("response ", response.Errors[0].Error);
  }
}
```

```
    console.log(
      `Device with id ${data.inputs.deviceId} was successfully updated in the
location tracking system. `,
      response,
    );
  } catch (error) {
    console.log("error ", error);
  }
};
```

- Einzelheiten zur API finden Sie [BatchUpdateDevicePosition](#) in der AWS SDK für JavaScript API-Referenz.

CalculateRoute

Das folgende Codebeispiel zeigt die Verwendung `CalculateRoute`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { fileURLToPath } from "node:url";
import {
  CalculateRouteCommand,
  ResourceNotFoundException,
  LocationClient,
} from "@aws-sdk/client-location";
import data from "./inputs.json" with { type: "json" };

const region = "eu-west-1";
const locationClient = new LocationClient({ region: region });

export const main = async () => {
  const routeCalcParams = {
    CalculatorName: `${data.inputs.calculatorName}`,
```

```
DeparturePosition: [-122.3321, 47.6062],
DestinationPosition: [-123.1216, 49.2827],
TravelMode: "Car",
DistanceUnit: "Kilometers",
};
try {
  const command = new CalculateRouteCommand(routeCalcParams);
  const response = await locationClient.send(command);

  console.log(
    "Successfully calculated route. The distance in kilometers is : ",
    response.Summary.Distance,
  );
} catch (caught) {
  if (caught instanceof ResourceNotFoundException) {
    console.error(
      `An conflict occurred: ${caught.message} \n Exiting program.`
    );
    return;
  }
}
};
```

- Einzelheiten zur API finden Sie [CalculateRoute](#) in der AWS SDK für JavaScript API-Referenz.

CreateGeofenceCollection

Das folgende Codebeispiel zeigt die Verwendung `CreateGeofenceCollection`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { fileURLToPath } from "node:url";
import {
```

```
    ConflictException,
    CreateGeofenceCollectionCommand,
    LocationClient,
  } from "@aws-sdk/client-location";
import data from "./inputs.json" with { type: "json" };

const region = "eu-west-1";

export const main = async () => {
  const geoFenceCollParams = {
    CollectionName: `${data.inputs.collectionName}`,
  };
  const locationClient = new LocationClient({ region: region });
  try {
    const command = new CreateGeofenceCollectionCommand(geoFenceCollParams);
    const response = await locationClient.send(command);
    console.log(
      "Collection created. Collection name is: ",
      response.CollectionName,
    );
  } catch (caught) {
    if (caught instanceof ConflictException) {
      console.error("A conflict occurred. Exiting program.");
      return;
    }
  }
};
```

- Einzelheiten zur API finden Sie [CreateGeofenceCollection](#) in der AWS SDK für JavaScript API-Referenz.

CreateMap

Das folgende Codebeispiel zeigt die Verwendung `CreateMap`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { fileURLToPath } from "node:url";
import { CreateMapCommand, LocationClient } from "@aws-sdk/client-location";
import data from "./inputs.json" with { type: "json" };

const region = "eu-west-1";

export const main = async () => {
  const CreateMapCommandInput = {
    MapName: `${data.inputs.mapName}`,
    Configuration: { style: "VectorEsriNavigation" },
  };
  const locationClient = new LocationClient({ region: region });
  try {
    const command = new CreateMapCommand(CreateMapCommandInput);
    const response = await locationClient.send(command);
    console.log("Map created. Map ARN is : ", response.MapArn);
  } catch (error) {
    console.error("Error creating map: ", error);
    throw error;
  }
};
```

- Einzelheiten zur API finden Sie [CreateMap](#) in der AWS SDK für JavaScript API-Referenz.

CreateRouteCalculator

Das folgende Codebeispiel zeigt die Verwendung `CreateRouteCalculator`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { fileURLToPath } from "node:url";
import {
  ConflictException,
  CreateRouteCalculatorCommand,
  LocationClient,
} from "@aws-sdk/client-location";
import data from "./inputs.json" with { type: "json" };

const region = "eu-west-1";
const locationClient = new LocationClient({ region: region });

export const main = async () => {
  const routeCalcParams = {
    CalculatorName: `${data.inputs.calculatorName}`,
    DataSource: "Esri",
  };
  try {
    const command = new CreateRouteCalculatorCommand(routeCalcParams);
    const response = await locationClient.send(command);

    console.log(
      "Route calculator created successfully. Calculator name is ",
      response.CalculatorName,
    );
  } catch (caught) {
    if (caught instanceof ConflictException) {
      console.error(
        `An conflict occurred: ${caught.message} \n Exiting program.`
      );
      return;
    }
  }
};
```

- Einzelheiten zur API finden Sie [CreateRouteCalculator](#) in der AWS SDK für JavaScript API-Referenz.

CreateTracker

Das folgende Codebeispiel zeigt die Verwendung `CreateTracker`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { fileURLToPath } from "node:url";
import { CreateTrackerCommand, LocationClient } from "@aws-sdk/client-location";
import data from "./inputs.json" with { type: "json" };

const region = "eu-west-1";

export const main = async () => {
  const createTrackerParams = {
    TrackerName: `${data.inputs.trackerName}`,
  };
  const locationClient = new LocationClient({ region: region });
  try {
    const command = new CreateTrackerCommand(createTrackerParams);
    const response = await locationClient.send(command);
    //state.trackerName - response.TrackerName;
    console.log("Tracker created. Tracker name is : ", response.TrackerName);
  } catch (error) {
    console.error("Error creating map: ", error);
    throw error;
  }
};
```

- Einzelheiten zur API finden Sie [CreateTracker](#) in der AWS SDK für JavaScript API-Referenz.

DeleteGeofenceCollection

Das folgende Codebeispiel zeigt die Verwendung `DeleteGeofenceCollection`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { fileURLToPath } from "node:url";
import {
  DeleteGeofenceCollectionCommand,
  LocationClient,
  ResourceNotFoundException,
} from "@aws-sdk/client-location";
import data from "../inputs.json" with { type: "json" };

const region = "eu-west-1";

export const main = async () => {
  const deleteGeofenceCollParams = {
    CollectionName: `${data.inputs.collectionName}`,
  };
  const locationClient = new LocationClient({ region: region });
  try {
    const command = new DeleteGeofenceCollectionCommand(
      deleteGeofenceCollParams,
    );
    const response = await locationClient.send(command);
    console.log("Collection deleted.");
  } catch (caught) {
    if (caught instanceof ResourceNotFoundException) {
      console.error(
        `${data.inputs.collectionName} Geofence collection not found.`,
      );
      return;
    }
  }
}
```



```
}  
};
```

- Einzelheiten zur API finden Sie [DeleteGeofenceCollection](#) in der AWS SDK für JavaScript API-Referenz.

DeleteMap

Das folgende Codebeispiel zeigt die Verwendung `DeleteMap`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { fileURLToPath } from "node:url";  
import {  
  DeleteMapCommand,  
  LocationClient,  
  ResourceNotFoundException,  
} from "@aws-sdk/client-location";  
import data from "./inputs.json" with { type: "json" };  
  
const region = "eu-west-1";  
  
export const main = async () => {  
  const deleteMapParams = {  
    MapName: `${data.inputs.mapName}`,  
  };  
  try {  
    const locationClient = new LocationClient({ region: region });  
    const command = new DeleteMapCommand(deleteMapParams);  
    const response = await locationClient.send(command);  
    console.log("Map deleted.");  
  } catch (caught) {  
    if (caught instanceof ResourceNotFoundException) {
```

```
        console.error(`${data.inputs.mapName} map not found.`);
        return;
    }
}
};
```

- Einzelheiten zur API finden Sie [DeleteMap](#) in der AWS SDK für JavaScript API-Referenz.

DeleteRouteCalculator

Das folgende Codebeispiel zeigt die Verwendung `DeleteRouteCalculator`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { fileURLToPath } from "node:url";
import {
    DeleteRouteCalculatorCommand,
    LocationClient,
    ResourceNotFoundException,
} from "@aws-sdk/client-location";
import data from "../inputs.json" with { type: "json" };

const region = "eu-west-1";

export const main = async () => {
    const deleteRouteCalculatorParams = {
        CalculatorName: `${data.inputs.calculatorName}`,
    };
    try {
        const locationClient = new LocationClient({ region: region });
        const command = new DeleteRouteCalculatorCommand(
            deleteRouteCalculatorParams,
        );
        const response = await locationClient.send(command);
```

```

    console.log("Route calculator deleted.");
  } catch (caught) {
    if (caught instanceof ResourceNotFoundException) {
      console.error(
        `${data.inputs.calculatorName} route calculator not found.`
      );
      return;
    }
  }
};

```

- Einzelheiten zur API finden Sie [DeleteRouteCalculator](#) in der AWS SDK für JavaScript API-Referenz.

DeleteTracker

Das folgende Codebeispiel zeigt die Verwendung `DeleteTracker`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```

import { fileURLToPath } from "node:url";
import {
  DeleteTrackerCommand,
  LocationClient,
  ResourceNotFoundException,
} from "@aws-sdk/client-location";
import data from "./inputs.json" with { type: "json" };

const region = "eu-west-1";

export const main = async () => {
  const deleteTrackerParams = {
    TrackerName: `${data.inputs.trackerName}`,

```

```
};
try {
  const locationClient = new LocationClient({ region: region });
  const command = new DeleteTrackerCommand(deleteTrackerParams);
  const response = await locationClient.send(command);
  console.log("Tracker deleted.");
} catch (caught) {
  if (caught instanceof ResourceNotFoundException) {
    console.error(`${data.inputs.trackerName} tracker not found.`);
    return;
  }
}
};
```

- Einzelheiten zur API finden Sie [DeleteTracker](#) in der AWS SDK für JavaScript API-Referenz.

GetDevicePosition

Das folgende Codebeispiel zeigt die Verwendung `GetDevicePosition`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { fileURLToPath } from "node:url";
import {
  GetDevicePositionCommand,
  LocationClient,
  ResourceNotFoundException,
} from "@aws-sdk/client-location";
import data from "../inputs.json" with { type: "json" };

const region = "eu-west-1";

export const main = async () => {
```

```
const locationClient = new LocationClient({ region: region });
const deviceId = `${data.inputs.deviceId}`;
const trackerName = `${data.inputs.trackerName}`;

const devicePositionParams = {
  DeviceId: deviceId,
  TrackerName: trackerName,
};
try {
  const command = new GetDevicePositionCommand(devicePositionParams);
  const response = await locationClient.send(command);
  //state.position = response.position;
  console.log("Successfully fetched device position: ", response);
} catch (error) {
  console.log("Error ", error);
  /* if (caught instanceof ResourceNotFoundException) {
    console.error(
      `The resource was not found: ${caught.message} \n Exiting program.`
    );
  } else {
    `An unexpected error error occurred: ${caught.message} \n Exiting program.`;
  }
  return;*/
}
};
```

- Einzelheiten zur API finden Sie [GetDevicePosition](#) in der AWS SDK für JavaScript API-Referenz.

PutGeofence

Das folgende Codebeispiel zeigt die Verwendung PutGeofence.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel](#) einrichten und ausführen.

```

import { fileURLToPath } from "node:url";
import {
  PutGeofenceCommand,
  LocationClient,
  ValidationException,
} from "@aws-sdk/client-location";
import data from "./inputs.json" with { type: "json" };

const region = "eu-west-1";
const locationClient = new LocationClient({ region: region });
export const main = async () => {
  const geoFenceGeoParams = {
    CollectionName: `${data.inputs.collectionName}`,
    GeofenceId: `${data.inputs.geoId}`,
    Geometry: {
      Polygon: [
        [
          [-122.3381, 47.6101],
          [-122.3281, 47.6101],
          [-122.3281, 47.6201],
          [-122.3381, 47.6201],
          [-122.3381, 47.6101],
        ],
      ],
    },
  };
  try {
    const command = new PutGeofenceCommand(geoFenceGeoParams);
    const response = await locationClient.send(command);
    console.log("GeoFence created. GeoFence ID is: ", response.GeofenceId);
  } catch (error) {
    console.error(
      `A validation error occurred while creating geofence: ${error} \n Exiting
program.`
    );
    return;
  }
};

```

- Einzelheiten zur API finden Sie [PutGeofence](#) in der AWS SDK für JavaScript API-Referenz.

Amazon MSK-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie AWS SDK für JavaScript (v3) mit Amazon MSK verwenden.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kodex finden.

Themen

- [Serverless-Beispiele](#)

Serverless-Beispiele

Aufrufen einer Lambda-Funktion über einen Amazon-MSK-Auslöser

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein Ereignis empfängt, das durch den Empfang von Datensätzen von einem Amazon MSK-Cluster ausgelöst wird. Die Funktion ruft die MSK-Nutzdaten ab und protokolliert den Inhalt des Datensatzes.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Ein Amazon MSK-Ereignis mit Lambda verwenden. JavaScript

```
exports.handler = async (event) => {
  // Iterate through keys
  for (let key in event.records) {
    console.log('Key: ', key)
    // Iterate through records
    event.records[key].map((record) => {
      console.log('Record: ', record)
      // Decode base64
      const msg = Buffer.from(record.value, 'base64').toString()
    })
  }
}
```

```
        console.log('Message:', msg)
    })
  }
}
```

Ein Amazon MSK-Ereignis mit Lambda verwenden. TypeScript

```
import { MSKEvent, Context } from "aws-lambda";
import { Buffer } from "buffer";
import { Logger } from "@aws-lambda-powertools/logger";

const logger = new Logger({
  logLevel: "INFO",
  serviceName: "msk-handler-sample",
});

export const handler = async (
  event: MSKEvent,
  context: Context
): Promise<void> => {
  for (const [topic, topicRecords] of Object.entries(event.records)) {
    logger.info(`Processing key: ${topic}`);

    // Process each record in the partition
    for (const record of topicRecords) {
      try {
        // Decode the message value from base64
        const decodedMessage = Buffer.from(record.value, 'base64').toString();

        logger.info({
          message: decodedMessage
        });
      }
      catch (error) {
        logger.error('Error processing event', { error });
        throw error;
      }
    }
  }
}
```


Amazon Personalize Personalize-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK für JavaScript (v3) mit Amazon Personalize Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kodex finden.

Themen

- [Aktionen](#)

Aktionen

CreateBatchInferenceJob

Das folgende Codebeispiel zeigt, wie Sie `CreateBatchInferenceJob`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
// Get service clients module and commands using ES6 syntax.
import { CreateBatchInferenceJobCommand } from "@aws-sdk/client-personalize";
import { personalizeClient } from "../libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the batch inference job's parameters.
```

```
export const createBatchInferenceJobParam = {
  jobName: "JOB_NAME",
  jobInput: {
    s3DataSource: {
      path: "INPUT_PATH",
    },
  },
  jobOutput: {
    s3DataDestination: {
      path: "OUTPUT_PATH",
    },
  },
  roleArn: "ROLE_ARN",
  solutionVersionArn: "SOLUTION_VERSION_ARN",
  numResults: 20,
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(
      new CreateBatchInferenceJobCommand(createBatchInferenceJobParam),
    );
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};

run();
```

- Einzelheiten zur API finden Sie [CreateBatchInferenceJob](#) in der AWS SDK für JavaScript API-Referenz.

CreateBatchSegmentJob

Das folgende Codebeispiel zeigt die Verwendung `CreateBatchSegmentJob`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
// Get service clients module and commands using ES6 syntax.
import { CreateBatchSegmentJobCommand } from "@aws-sdk/client-personalize";
import { personalizeClient } from "../libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the batch segment job's parameters.

export const createBatchSegmentJobParam = {
  jobName: "NAME",
  jobInput: {
    s3DataSource: {
      path: "INPUT_PATH",
    },
  },
  jobOutput: {
    s3DataDestination: {
      path: "OUTPUT_PATH",
    },
  },
  roleArn: "ROLE_ARN",
  solutionVersionArn: "SOLUTION_VERSION_ARN",
  numResults: 20,
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(
      new CreateBatchSegmentJobCommand(createBatchSegmentJobParam),
    );
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
```

```
    console.log("Error", err);
  }
};
run();
```

- Einzelheiten zur API finden Sie [CreateBatchSegmentJob](#) in der AWS SDK für JavaScript API-Referenz.

CreateCampaign

Das folgende Codebeispiel zeigt die Verwendung `CreateCampaign`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
// Get service clients module and commands using ES6 syntax.

import { CreateCampaignCommand } from "@aws-sdk/client-personalize";
import { personalizeClient } from "../libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the campaign's parameters.
export const createCampaignParam = {
  solutionVersionArn: "SOLUTION_VERSION_ARN" /* required */,
  name: "NAME" /* required */,
  minProvisionedTPS: 1 /* optional integer */,
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(
      new CreateCampaignCommand(createCampaignParam),
    );
  }
```

```
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Einzelheiten zur API finden Sie [CreateCampaign](#) in der AWS SDK für JavaScript API-Referenz.

CreateDataset

Das folgende Codebeispiel zeigt die Verwendung `CreateDataset`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
// Get service clients module and commands using ES6 syntax.
import { CreateDatasetCommand } from "@aws-sdk/client-personalize";
import { personalizeClient } from "../libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the dataset's parameters.
export const createDatasetParam = {
  datasetGroupArn: "DATASET_GROUP_ARN" /* required */,
  datasetType: "DATASET_TYPE" /* required */,
  name: "NAME" /* required */,
  schemaArn: "SCHEMA_ARN" /* required */,
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(
      new CreateDatasetCommand(createDatasetParam),
```

```
    );  
    console.log("Success", response);  
    return response; // For unit tests.  
  } catch (err) {  
    console.log("Error", err);  
  }  
};  
run();
```

- Einzelheiten zur API finden Sie [CreateDataset](#) in der AWS SDK für JavaScript API-Referenz.

CreateDatasetExportJob

Das folgende Codebeispiel zeigt die Verwendung `CreateDatasetExportJob`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
// Get service clients module and commands using ES6 syntax.  
import { CreateDatasetExportJobCommand } from "@aws-sdk/client-personalize";  
import { personalizeClient } from "../libs/personalizeClients.js";  
  
// Or, create the client here.  
// const personalizeClient = new PersonalizeClient({ region: "REGION"});  
  
// Set the export job parameters.  
export const datasetExportJobParam = {  
  datasetArn: "DATASET_ARN" /* required */,  
  jobOutput: {  
    s3DataDestination: {  
      path: "S3_DESTINATION_PATH" /* required */,  
      //kmsKeyArn: 'ARN' /* include if your bucket uses AWS KMS for encryption  
    },  
  },  
  jobName: "NAME" /* required */,  
  roleArn: "ROLE_ARN" /* required */,  
};
```

```
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(
      new CreateDatasetExportJobCommand(datasetExportJobParam),
    );
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};

run();
```

- Einzelheiten zur API finden Sie [CreateDatasetExportJob](#) in der AWS SDK für JavaScript API-Referenz.

CreateDatasetGroup

Das folgende Codebeispiel zeigt die Verwendung `CreateDatasetGroup`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
// Get service clients module and commands using ES6 syntax.

import { CreateDatasetGroupCommand } from "@aws-sdk/client-personalize";
import { personalizeClient } from "../libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the dataset group parameters.
export const createDatasetGroupParam = {
```

```
    name: "NAME" /* required */,
  };

export const run = async (createDatasetGroupParam) => {
  try {
    const response = await personalizeClient.send(
      new CreateDatasetGroupCommand(createDatasetGroupParam),
    );
    console.log("Success", response);
    return "Run successfully"; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};

run(createDatasetGroupParam);
```

Erstellen Sie eine Domain-Datensatzgruppe.

```
// Get service clients module and commands using ES6 syntax.
import { CreateDatasetGroupCommand } from "@aws-sdk/client-personalize";
import { personalizeClient } from "../libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the domain dataset group parameters.
export const domainDatasetGroupParams = {
  name: "NAME" /* required */,
  domain:
    "DOMAIN" /* required for a domain dsd, specify ECOMMERCE or VIDEO_ON_DEMAND */,
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(
      new CreateDatasetGroupCommand(domainDatasetGroupParams),
    );
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
}
```



```
};  
run();
```

- Einzelheiten zur API finden Sie [CreateDatasetGroup](#) in der AWS SDK für JavaScript API-Referenz.

CreateDatasetImportJob

Das folgende Codebeispiel zeigt die Verwendung `CreateDatasetImportJob`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
// Get service clients module and commands using ES6 syntax.  
import { CreateDatasetImportJobCommand } from "@aws-sdk/client-personalize";  
import { personalizeClient } from "../libs/personalizeClients.js";  
  
// Or, create the client here.  
// const personalizeClient = new PersonalizeClient({ region: "REGION"});  
  
// Set the dataset import job parameters.  
export const datasetImportJobParam = {  
  datasetArn: "DATASET_ARN" /* required */,  
  dataSource: {  
    /* required */  
    dataLocation: "S3_PATH",  
  },  
  jobName: "NAME" /* required */,  
  roleArn: "ROLE_ARN" /* required */,  
};  
  
export const run = async () => {  
  try {  
    const response = await personalizeClient.send(  
      new CreateDatasetImportJobCommand(datasetImportJobParam),  
    );  
  }  
}
```

```
    );  
    console.log("Success", response);  
    return response; // For unit tests.  
  } catch (err) {  
    console.log("Error", err);  
  }  
};  
run();
```

- Einzelheiten zur API finden Sie [CreateDatasetImportJob](#) in der AWS SDK für JavaScript API-Referenz.

CreateEventTracker

Das folgende Codebeispiel zeigt die Verwendung `CreateEventTracker`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
// Get service clients module and commands using ES6 syntax.  
import { CreateEventTrackerCommand } from "@aws-sdk/client-personalize";  
import { personalizeClient } from "../libs/personalizeClients.js";  
  
// Or, create the client here.  
// const personalizeClient = new PersonalizeClient({ region: "REGION"});  
  
// Set the event tracker's parameters.  
export const createEventTrackerParam = {  
  datasetGroupArn: "DATASET_GROUP_ARN" /* required */,  
  name: "NAME" /* required */,  
};  
  
export const run = async () => {  
  try {  
    const response = await personalizeClient.send(  

```

```

        new CreateEventTrackerCommand(createEventTrackerParam),
    );
    console.log("Success", response);
    return response; // For unit tests.
} catch (err) {
    console.log("Error", err);
}
};
run();

```

- Einzelheiten zur API finden Sie [CreateEventTracker](#) in der AWS SDK für JavaScript API-Referenz.

CreateFilter

Das folgende Codebeispiel zeigt die Verwendung `CreateFilter`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```

// Get service clients module and commands using ES6 syntax.
import { CreateFilterCommand } from "@aws-sdk/client-personalize";
import { personalizeClient } from "../libs/personalizeClients.js";
// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the filter's parameters.
export const createFilterParam = {
    datasetGroupArn: "DATASET_GROUP_ARN" /* required */,
    name: "NAME" /* required */,
    filterExpression: "FILTER_EXPRESSION" /*required */,
};

export const run = async () => {
    try {

```

```
const response = await personalizeClient.send(  
  new CreateFilterCommand(createFilterParam),  
);  
console.log("Success", response);  
return response; // For unit tests.  
} catch (err) {  
  console.log("Error", err);  
}  
};  
run();
```

- Einzelheiten zur API finden Sie [CreateFilter](#) in der AWS SDK für JavaScript API-Referenz.

CreateRecommender

Das folgende Codebeispiel zeigt die Verwendung `CreateRecommender`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
// Get service clients module and commands using ES6 syntax.  
import { CreateRecommenderCommand } from "@aws-sdk/client-personalize";  
import { personalizeClient } from "../libs/personalizeClients.js";  
  
// Or, create the client here.  
// const personalizeClient = new PersonalizeClient({ region: "REGION"});  
  
// Set the recommender's parameters.  
export const createRecommenderParam = {  
  name: "NAME" /* required */,  
  recipeArn: "RECIPE_ARN" /* required */,  
  datasetGroupArn: "DATASET_GROUP_ARN" /* required */,  
};  
  
export const run = async () => {  
  try {
```

```
const response = await personalizeClient.send(
  new CreateRecommenderCommand(createRecommenderParam),
);
console.log("Success", response);
return response; // For unit tests.
} catch (err) {
  console.log("Error", err);
}
};
run();
```

- Einzelheiten zur API finden Sie [CreateRecommender](#) in der AWS SDK für JavaScript API-Referenz.

CreateSchema

Das folgende Codebeispiel zeigt die Verwendung `CreateSchema`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
// Get service clients module and commands using ES6 syntax.
import { CreateSchemaCommand } from "@aws-sdk/client-personalize";
import { personalizeClient } from "../libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

import fs from "node:fs";

const schemaFilePath = "SCHEMA_PATH";
let mySchema = "";

try {
  mySchema = fs.readFileSync(schemaFilePath).toString();
}
```

```
} catch (err) {
  mySchema = "TEST"; // For unit tests.
}
// Set the schema parameters.
export const createSchemaParam = {
  name: "NAME" /* required */,
  schema: mySchema /* required */,
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(
      new CreateSchemaCommand(createSchemaParam),
    );
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Erstellen Sie ein Schema mit einer Domain.

```
// Get service clients module and commands using ES6 syntax.
import { CreateSchemaCommand } from "@aws-sdk/client-personalize";
import { personalizeClient } from "../libs/personalizeClients.js";

// Or, create the client here.
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

import fs from "node:fs";

const schemaFilePath = "SCHEMA_PATH";
let mySchema = "";

try {
  mySchema = fs.readFileSync(schemaFilePath).toString();
} catch (err) {
  mySchema = "TEST"; // for unit tests.
}
```

```
// Set the domain schema parameters.
export const createDomainSchemaParam = {
  name: "NAME" /* required */,
  schema: mySchema /* required */,
  domain:
    "DOMAIN" /* required for a domain dataset group, specify ECOMMERCE or
    VIDEO_ON_DEMAND */,
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(
      new CreateSchemaCommand(createDomainSchemaParam),
    );
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Einzelheiten zur API finden Sie [CreateSchema](#) in der AWS SDK für JavaScript API-Referenz.

CreateSolution

Das folgende Codebeispiel zeigt die Verwendung `CreateSolution`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
// Get service clients module and commands using ES6 syntax.
import { CreateSolutionCommand } from "@aws-sdk/client-personalize";
import { personalizeClient } from "../libs/personalizeClients.js";
// Or, create the client here.
```

```
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the solution parameters.
export const createSolutionParam = {
  datasetGroupArn: "DATASET_GROUP_ARN" /* required */,
  recipeArn: "RECIPE_ARN" /* required */,
  name: "NAME" /* required */,
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(
      new CreateSolutionCommand(createSolutionParam),
    );
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Einzelheiten zur API finden Sie [CreateSolution](#) in der AWS SDK für JavaScript API-Referenz.

CreateSolutionVersion

Das folgende Codebeispiel zeigt die Verwendung `CreateSolutionVersion`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
// Get service clients module and commands using ES6 syntax.
import { CreateSolutionVersionCommand } from "@aws-sdk/client-personalize";
import { personalizeClient } from "../libs/personalizeClients.js";
// Or, create the client here.
```



```
// const personalizeClient = new PersonalizeClient({ region: "REGION"});

// Set the solution version parameters.
export const solutionVersionParam = {
  solutionArn: "SOLUTION_ARN" /* required */,
};

export const run = async () => {
  try {
    const response = await personalizeClient.send(
      new CreateSolutionVersionCommand(solutionVersionParam),
    );
    console.log("Success", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Einzelheiten zur API finden Sie [CreateSolutionVersion](#) in der AWS SDK für JavaScript API-Referenz.

Beispiele für Amazon Personalize Events mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie AWS SDK für JavaScript (v3) mit Amazon Personalize Events verwenden.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kodex finden.

Themen

- [Aktionen](#)

Aktionen

PutEvents

Das folgende Codebeispiel zeigt die Verwendung PutEvents von.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
// Get service clients module and commands using ES6 syntax.
import { PutEventsCommand } from "@aws-sdk/client-personalize-events";
import { personalizeEventsClient } from "../libs/personalizeClients.js";
// Or, create the client here.
// const personalizeEventsClient = new PersonalizeEventsClient({ region: "REGION"});

// Convert your UNIX timestamp to a Date.
const sentAtDate = new Date(1613443801 * 1000); // 1613443801 is a testing value.
Replace it with your sentAt timestamp in UNIX format.

// Set put events parameters.
const putEventsParam = {
  eventList: [
    /* required */
    {
      eventType: "EVENT_TYPE" /* required */,
      sentAt: sentAtDate /* required, must be a Date with js */,
      eventId: "EVENT_ID" /* optional */,
      itemId: "ITEM_ID" /* optional */,
    },
  ],
  sessionId: "SESSION_ID" /* required */,
  trackingId: "TRACKING_ID" /* required */,
  userId: "USER_ID" /* required */,
};
export const run = async () => {
  try {
    const response = await personalizeEventsClient.send(
```

```
        new PutEventsCommand(putEventsParam),
    );
    console.log("Success!", response);
    return response; // For unit tests.
} catch (err) {
    console.log("Error", err);
}
};
run();
```

- Einzelheiten zur API finden Sie [PutEvents](#) in der AWS SDK für JavaScript API-Referenz.

PutItems

Das folgende Codebeispiel zeigt die Verwendung `PutItems`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
// Get service clients module and commands using ES6 syntax.
import { PutItemsCommand } from "@aws-sdk/client-personalize-events";
import { personalizeEventsClient } from "../libs/personalizeClients.js";
// Or, create the client here.
// const personalizeEventsClient = new PersonalizeEventsClient({ region: "REGION"});

// Set the put items parameters. For string properties and values, use the \
character to escape quotes.
const putItemsParam = {
  datasetArn: "DATASET_ARN" /* required */,
  items: [
    /* required */
    {
      itemId: "ITEM_ID" /* required */,
      properties:
        '{"PROPERTY1_NAME": "PROPERTY1_VALUE", "PROPERTY2_NAME": "PROPERTY2_VALUE",
        "PROPERTY3_NAME": "PROPERTY3_VALUE"}' /* optional */,
```

```

    },
  ],
};
export const run = async () => {
  try {
    const response = await personalizeEventsClient.send(
      new PutItemsCommand(putItemsParam),
    );
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();

```

- Einzelheiten zur API finden Sie [PutItems](#) in der AWS SDK für JavaScript API-Referenz.

PutUsers

Das folgende Codebeispiel zeigt die Verwendung `PutUsers`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```

// Get service clients module and commands using ES6 syntax.
import { PutUsersCommand } from "@aws-sdk/client-personalize-events";
import { personalizeEventsClient } from "../libs/personalizeClients.js";
// Or, create the client here.
// const personalizeEventsClient = new PersonalizeEventsClient({ region: "REGION"});

// Set the put users parameters. For string properties and values, use the \
character to escape quotes.
const putUsersParam = {
  datasetArn: "DATASET_ARN",

```

```
users: [
  {
    userId: "USER_ID",
    properties: '{"PROPERTY1_NAME": "PROPERTY1_VALUE"}',
  },
],
};
export const run = async () => {
  try {
    const response = await personalizeEventsClient.send(
      new PutUsersCommand(putUsersParam),
    );
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Einzelheiten zur API finden Sie [PutUsers](#) in der AWS SDK für JavaScript API-Referenz.

Amazon Personalize Runtime-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie AWS SDK für JavaScript (v3) mit Amazon Personalize Runtime verwenden.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kodex finden.

Themen

- [Aktionen](#)

Aktionen

GetPersonalizedRanking

Das folgende Codebeispiel zeigt, wie Sie `GetPersonalizedRanking`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
// Get service clients module and commands using ES6 syntax.
import { GetPersonalizedRankingCommand } from "@aws-sdk/client-personalize-runtime";
import { personalizeRuntimeClient } from "../libs/personalizeClients.js";
// Or, create the client here.
// const personalizeRuntimeClient = new PersonalizeRuntimeClient({ region:
  "REGION"});

// Set the ranking request parameters.
export const getPersonalizedRankingParam = {
  campaignArn: "CAMPAIGN_ARN" /* required */,
  userId: "USER_ID" /* required */,
  inputList: ["ITEM_ID_1", "ITEM_ID_2", "ITEM_ID_3", "ITEM_ID_4"],
};

export const run = async () => {
  try {
    const response = await personalizeRuntimeClient.send(
      new GetPersonalizedRankingCommand(getPersonalizedRankingParam),
    );
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};

run();
```

- Einzelheiten zur API finden Sie [GetPersonalizedRanking](#) in der AWS SDK für JavaScript API-Referenz.

GetRecommendations

Das folgende Codebeispiel zeigt die Verwendung `GetRecommendations`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
// Get service clients module and commands using ES6 syntax.
import { GetRecommendationsCommand } from "@aws-sdk/client-personalize-runtime";

import { personalizeRuntimeClient } from "../libs/personalizeClients.js";
// Or, create the client here.
// const personalizeRuntimeClient = new PersonalizeRuntimeClient({ region:
  "REGION"});

// Set the recommendation request parameters.
export const getRecommendationsParam = {
  campaignArn: "CAMPAIGN_ARN" /* required */,
  userId: "USER_ID" /* required */,
  numResults: 15 /* optional */,
};

export const run = async () => {
  try {
    const response = await personalizeRuntimeClient.send(
      new GetRecommendationsCommand(getRecommendationsParam),
    );
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};

run();
```

Holen Sie sich eine Empfehlung mit einem Filter (benutzerdefinierte Datensatzgruppe).

```
// Get service clients module and commands using ES6 syntax.
import { GetRecommendationsCommand } from "@aws-sdk/client-personalize-runtime";
import { personalizeRuntimeClient } from "../libs/personalizeClients.js";
// Or, create the client here.
// const personalizeRuntimeClient = new PersonalizeRuntimeClient({ region:
  "REGION"});

// Set the recommendation request parameters.
export const getRecommendationsParam = {
  recommenderArn: "RECOMMENDER_ARN" /* required */,
  userId: "USER_ID" /* required */,
  numResults: 15 /* optional */,
};

export const run = async () => {
  try {
    const response = await personalizeRuntimeClient.send(
      new GetRecommendationsCommand(getRecommendationsParam),
    );
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

Holen Sie sich gefilterte Empfehlungen von einem Empfehler, der in einer Domain-Datensatzgruppe erstellt wurde.

```
// Get service clients module and commands using ES6 syntax.
import { GetRecommendationsCommand } from "@aws-sdk/client-personalize-runtime";
import { personalizeRuntimeClient } from "../libs/personalizeClients.js";
// Or, create the client here:
// const personalizeRuntimeClient = new PersonalizeRuntimeClient({ region:
  "REGION"});
```



```
// Set recommendation request parameters.
export const getRecommendationsParam = {
  campaignArn: "CAMPAIGN_ARN" /* required */,
  userId: "USER_ID" /* required */,
  numResults: 15 /* optional */,
  filterArn: "FILTER_ARN" /* required to filter recommendations */,
  filterValues: {
    PROPERTY:
      '"VALUE"' /* Only required if your filter has a placeholder parameter */,
  },
};

export const run = async () => {
  try {
    const response = await personalizeRuntimeClient.send(
      new GetRecommendationsCommand(getRecommendationsParam),
    );
    console.log("Success!", response);
    return response; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};

run();
```

- Einzelheiten zur API finden Sie [GetRecommendations](#) in der AWS SDK für JavaScript API-Referenz.

Amazon Pinpoint Pinpoint-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK für JavaScript (v3) mit Amazon Pinpoint Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kodex finden.

Themen

- [Aktionen](#)

Aktionen

SendMessage

Das folgende Codebeispiel zeigt die Verwendung `SendMessage` von.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie den Client in einem separaten Modul und exportieren Sie ihn.

```
import { PinpointClient } from "@aws-sdk/client-pinpoint";
// Set the AWS Region.
const REGION = "us-east-1";
export const pinClient = new PinpointClient({ region: REGION });
```

Senden Sie eine E-Mail-Nachricht.

```
// Import required AWS SDK clients and commands for Node.js
import { SendMessageCommand } from "@aws-sdk/client-pinpoint";
import { pinClient } from "../libs/pinClient.js";

// The FromAddress must be verified in SES.
const fromAddress = "FROM_ADDRESS";
const toAddress = "TO_ADDRESS";
const projectId = "PINPOINT_PROJECT_ID";

// The subject line of the email.
const subject = "Amazon Pinpoint Test (AWS SDK for JavaScript in Node.js)";

// The email body for recipients with non-HTML email clients.
const body_text = `Amazon Pinpoint Test (SDK for JavaScript in Node.js)
-----`
```

This email was sent with Amazon Pinpoint using the AWS SDK for JavaScript in Node.js.

For more information, see <https://aws.amazon.com/sdk-for-node-js/>;

// The body of the email for recipients whose email clients support HTML content.

```
const body_html = `<html>
```

```
<head></head>
```

```
<body>
```

```
  <h1>Amazon Pinpoint Test (SDK for JavaScript in Node.js)</h1>
```

```
  <p>This email was sent with
```

```
    <a href='https://aws.amazon.com/pinpoint/'>the Amazon Pinpoint Email API</a>  
using the
```

```
  <a href='https://aws.amazon.com/sdk-for-node-js/'>
```

```
    AWS SDK for JavaScript in Node.js</a>.</p>
```

```
</body>
```

```
</html>`;
```

// The character encoding for the subject line and message body of the email.

```
const charset = "UTF-8";
```

```
const params = {
```

```
  ApplicationId: projectId,
```

```
  MessageRequest: {
```

```
    Addresses: {
```

```
      [toAddress]: {
```

```
        ChannelType: "EMAIL",
```

```
      },
```

```
    },
```

```
    MessageConfiguration: {
```

```
      EmailMessage: {
```

```
        FromAddress: fromAddress,
```

```
        SimpleEmail: {
```

```
          Subject: {
```

```
            Charset: charset,
```

```
            Data: subject,
```

```
          },
```

```
          HtmlPart: {
```

```
            Charset: charset,
```

```
            Data: body_html,
```

```
          },
```

```
          TextPart: {
```

```
            Charset: charset,
```

```
            Data: body_text,
```

```
          },
```

```

        },
      },
    },
  },
};

const run = async () => {
  try {
    const { MessageResponse } = await pinClient.send(
      new SendMessagesCommand(params),
    );

    if (!MessageResponse) {
      throw new Error("No message response.");
    }

    if (!MessageResponse.Result) {
      throw new Error("No message result.");
    }

    const recipientResult = MessageResponse.Result[toAddress];

    if (recipientResult.StatusCode !== 200) {
      throw new Error(recipientResult.StatusMessage);
    }
    console.log(recipientResult.MessageId);
  } catch (err) {
    console.log(err.message);
  }
};

run();

```

Senden Sie eine SMS-Nachricht.

```

// Import required AWS SDK clients and commands for Node.js
import { SendMessagesCommand } from "@aws-sdk/client-pinpoint";
import { pinClient } from "../libs/pinClient.js";

/* The phone number or short code to send the message from. The phone number
or short code that you specify has to be associated with your Amazon Pinpoint

```

```
account. For best results, specify long codes in E.164 format. */
const originationNumber = "SENDER_NUMBER"; //e.g., +1XXXXXXXXXX

// The recipient's phone number. For best results, you should specify the phone
// number in E.164 format.
const destinationNumber = "RECEIVER_NUMBER"; //e.g., +1XXXXXXXXXX

// The content of the SMS message.
const message =
  "This message was sent through Amazon Pinpoint " +
  "using the AWS SDK for JavaScript in Node.js. Reply STOP to " +
  "opt out.";

/*The Amazon Pinpoint project/application ID to use when you send this message.
Make sure that the SMS channel is enabled for the project or application
that you choose.*/
const projectId = "PINPOINT_PROJECT_ID"; //e.g., XXXXXXXX66e4e9986478cXXXXXXXXXX

/* The type of SMS message that you want to send. If you plan to send
time-sensitive content, specify TRANSACTIONAL. If you plan to send
marketing-related content, specify PROMOTIONAL.*/
const messageType = "TRANSACTIONAL";

// The registered keyword associated with the originating short code.
const registeredKeyword = "myKeyword";

/* The sender ID to use when sending the message. Support for sender ID
// varies by country or region. For more information, see
https://docs.aws.amazon.com/pinpoint/latest/userguide/channels-sms-countries.html.*/

const senderId = "MySenderId";

// Specify the parameters to pass to the API.
const params = {
  ApplicationId: projectId,
  MessageRequest: {
    Addresses: {
      [destinationNumber]: {
        ChannelType: "SMS",
      },
    },
    MessageConfiguration: {
      SMSMessage: {
        Body: message,
```

```
        Keyword: registeredKeyword,
        MessageType: messageType,
        OriginationNumber: originationNumber,
        SenderId: senderId,
    },
},
};

const run = async () => {
    try {
        const data = await pinClient.send(new SendMessagesCommand(params));
        console.log(
            `Message sent!
${data.MessageResponse.Result[destinationNumber].StatusMessage}`,
        );
    } catch (err) {
        console.log(err);
    }
};

run();
```

- Einzelheiten zur API finden Sie [SendMessages](#) in der AWS SDK für JavaScript API-Referenz.

Amazon Polly Polly-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie AWS SDK für JavaScript (v3) mit Amazon Polly verwenden.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb eines Service aufrufen oder mit anderen AWS-Services kombinieren.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kodex finden.

Themen

- [Szenarien](#)

Szenarien

Erstellen einer Anwendung zum Analysieren von Kundenfeedback

Das folgende Codebeispiel zeigt, wie Sie eine Anwendung erstellen, die Kundenkommentarkarten analysiert, sie aus der ursprünglichen Sprache übersetzt, die Stimmung ermittelt und auf der Grundlage des übersetzten Texts eine Audiodatei generiert.

SDK für JavaScript (v3)

Diese Beispielanwendung analysiert und speichert Kundenfeedback-Karten. Sie ist auf die Anforderungen eines fiktiven Hotels in New York City zugeschnitten. Das Hotel erhält Feedback von Gästen in Form von physischen Kommentarkarten in verschiedenen Sprachen. Dieses Feedback wird über einen Webclient in die App hochgeladen. Nachdem ein Bild einer Kommentarkarte hochgeladen wurde, werden folgende Schritte ausgeführt:

- Der Text wird mithilfe von Amazon Textract aus dem Bild extrahiert.
- Amazon Comprehend ermittelt die Stimmung und die Sprache des extrahierten Textes.
- Der extrahierte Text wird mithilfe von Amazon Translate ins Englische übersetzt.
- Amazon Polly generiert auf der Grundlage des extrahierten Texts eine Audiodatei.

Die vollständige App kann mithilfe des AWS CDK bereitgestellt werden. Quellcode und Anweisungen zur Bereitstellung finden Sie im Projekt unter [GitHub](#). Die folgenden Auszüge zeigen, wie der innerhalb von Lambda-Funktionen verwendet AWS SDK für JavaScript wird.

```
import {
  ComprehendClient,
  DetectDominantLanguageCommand,
  DetectSentimentCommand,
} from "@aws-sdk/client-comprehend";

/**
 * Determine the language and sentiment of the extracted text.
 *
 * @param {{ source_text: string }} extractTextOutput
 */
export const handler = async (extractTextOutput) => {
  const comprehendClient = new ComprehendClient({});

  const detectDominantLanguageCommand = new DetectDominantLanguageCommand({
    Text: extractTextOutput.source_text,
```

```

});

// The source language is required for sentiment analysis and
// translation in the next step.
const { Languages } = await comprehendClient.send(
  detectDominantLanguageCommand,
);

const languageCode = Languages[0].LanguageCode;

const detectSentimentCommand = new DetectSentimentCommand({
  Text: extractTextOutput.source_text,
  LanguageCode: languageCode,
});

const { Sentiment } = await comprehendClient.send(detectSentimentCommand);

return {
  sentiment: Sentiment,
  language_code: languageCode,
};
};

```

```

import {
  DetectDocumentTextCommand,
  TextractClient,
} from "@aws-sdk/client-textract";

/**
 * Fetch the S3 object from the event and analyze it using Amazon Textract.
 *
 * @param {import("@types/aws-lambda").EventBridgeEvent<"Object Created">}
  eventBridgeS3Event
 */
export const handler = async (eventBridgeS3Event) => {
  const textractClient = new TextractClient();

  const detectDocumentTextCommand = new DetectDocumentTextCommand({
    Document: {
      S3Object: {
        Bucket: eventBridgeS3Event.bucket,
        Name: eventBridgeS3Event.object,
      },
    },
  },

```



```

    },
  });

  // Textract returns a list of blocks. A block can be a line, a page, word, etc.
  // Each block also contains geometry of the detected text.
  // For more information on the Block type, see https://docs.aws.amazon.com/
  // textract/latest/dg/API_Block.html.
  const { Blocks } = await textractClient.send(detectDocumentTextCommand);

  // For the purpose of this example, we are only interested in words.
  const extractedWords = Blocks.filter((b) => b.BlockType === "WORD").map(
    (b) => b.Text,
  );

  return extractedWords.join(" ");
};

```

```

import { PollyClient, SynthesizeSpeechCommand } from "@aws-sdk/client-polly";
import { S3Client } from "@aws-sdk/client-s3";
import { Upload } from "@aws-sdk/lib-storage";

/**
 * Synthesize an audio file from text.
 *
 * @param {{ bucket: string, translated_text: string, object: string }}
 * sourceDestinationConfig
 */
export const handler = async (sourceDestinationConfig) => {
  const pollyClient = new PollyClient({});

  const synthesizeSpeechCommand = new SynthesizeSpeechCommand({
    Engine: "neural",
    Text: sourceDestinationConfig.translated_text,
    VoiceId: "Ruth",
    OutputFormat: "mp3",
  });

  const { AudioStream } = await pollyClient.send(synthesizeSpeechCommand);

  const audioKey = `${sourceDestinationConfig.object}.mp3`;

  // Store the audio file in S3.
  const s3Client = new S3Client();

```

```
const upload = new Upload({
  client: s3Client,
  params: {
    Bucket: sourceDestinationConfig.bucket,
    Key: audioKey,
    Body: AudioStream,
    ContentType: "audio/mp3",
  },
});

await upload.done();
return audioKey;
};
```

```
import {
  TranslateClient,
  TranslateTextCommand,
} from "@aws-sdk/client-translate";

/**
 * Translate the extracted text to English.
 *
 * @param {{ extracted_text: string, source_language_code: string }}
  textAndSourceLanguage
 */
export const handler = async (textAndSourceLanguage) => {
  const translateClient = new TranslateClient({});

  const translateCommand = new TranslateTextCommand({
    SourceLanguageCode: textAndSourceLanguage.source_language_code,
    TargetLanguageCode: "en",
    Text: textAndSourceLanguage.extracted_text,
  });

  const { TranslatedText } = await translateClient.send(translateCommand);

  return { translated_text: TranslatedText };
};
```

In diesem Beispiel verwendete Dienste

- Amazon Comprehend
- Lambda

- Amazon Polly
- Amazon Textract
- Amazon Translate

Amazon RDS-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie AWS SDK für JavaScript (v3) mit Amazon RDS verwenden.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb eines Service aufrufen oder mit anderen AWS-Services kombinieren.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kodex finden.

Themen

- [Szenarien](#)
- [Serverless-Beispiele](#)

Szenarien

Erstellen eines Trackers für Aurora-Serverless-Arbeitsaufgaben

Das folgende Codebeispiel zeigt, wie eine Webanwendung erstellt wird, die Arbeitselemente in einer Datenbank von Amazon Aurora Serverless verfolgt und mithilfe von Amazon Simple Email Service (Amazon SES) Berichte sendet.

SDK für JavaScript (v3)

Zeigt, wie Sie mit AWS SDK für JavaScript (v3) eine Webanwendung erstellen, die Arbeitselemente in einer Amazon Aurora Datenbank verfolgt und Berichte mithilfe von Amazon Simple Email Service (Amazon SES) per E-Mail versendet. In diesem Beispiel wird ein mit React.js erstelltes Frontend verwendet, um mit einem Express-Node.js-Backend zu interagieren.

- Integrieren Sie eine React.js Webanwendung mit AWS-Services.
- Auflisten, hinzufügen und aktualisieren von Elementen in einer Aurora-Tabelle.
- Senden Sie einen E-Mail-Bericht über gefilterte Arbeitselemente mit Amazon SES.

- Stellen Sie Beispielressourcen mit dem mitgelieferten AWS CloudFormation Skript bereit und verwalten Sie sie.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

In diesem Beispiel verwendete Dienste

- Aurora
- Amazon RDS
- Amazon RDS Data Service
- Amazon SES

Serverless-Beispiele

Herstellen einer Verbindung mit einer Amazon-RDS-Datenbank in einer Lambda-Funktion

Die folgenden Codebeispiele veranschaulichen, wie eine Lambda-Funktion implementiert wird, die eine Verbindung zu einer RDS-Datenbank herstellt. Die Funktion stellt eine einfache Datenbankanfrage und gibt das Ergebnis zurück.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Herstellen einer Verbindung zu einer Amazon RDS-Datenbank in einer Lambda-Funktion mithilfe von JavaScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
/*
Node.js code here.
*/
// ES6+ example
import { Signer } from "@aws-sdk/rds-signer";
import mysql from 'mysql2/promise';
```

```
async function createAuthToken() {
  // Define connection authentication parameters
  const dbinfo = {

    hostname: process.env.ProxyHostName,
    port: process.env.Port,
    username: process.env.DBUserName,
    region: process.env.AWS_REGION,

  }

  // Create RDS Signer object
  const signer = new Signer(dbinfo);

  // Request authorization token from RDS, specifying the username
  const token = await signer.getAuthToken();
  return token;
}

async function dbOps() {

  // Obtain auth token
  const token = await createAuthToken();
  // Define connection configuration
  let connectionConfig = {
    host: process.env.ProxyHostName,
    user: process.env.DBUserName,
    password: token,
    database: process.env.DBName,
    ssl: 'Amazon RDS'
  }
  // Create the connection to the DB
  const conn = await mysql.createConnection(connectionConfig);
  // Obtain the result of the query
  const [res,] = await conn.execute('select ?+? as sum', [3, 2]);
  return res;

}

export const handler = async (event) => {
  // Execute database flow
  const result = await dbOps();
  // Return result
```

```
return {
  statusCode: 200,
  body: JSON.stringify("The selected sum is: " + result[0].sum)
};
```

Herstellen einer Verbindung zu einer Amazon RDS-Datenbank in einer Lambda-Funktion mithilfe von TypeScript.

```
import { Signer } from "@aws-sdk/rds-signer";
import mysql from 'mysql2/promise';

// RDS settings
// Using '!' (non-null assertion operator) to tell the TypeScript compiler that the
// DB settings are not null or undefined,
const proxy_host_name = process.env.PROXY_HOST_NAME!
const port = parseInt(process.env.PORT!)
const db_name = process.env.DB_NAME!
const db_user_name = process.env.DB_USER_NAME!
const aws_region = process.env.AWS_REGION!

async function createAuthToken(): Promise<string> {

  // Create RDS Signer object
  const signer = new Signer({
    hostname: proxy_host_name,
    port: port,
    region: aws_region,
    username: db_user_name
  });

  // Request authorization token from RDS, specifying the username
  const token = await signer.getAuthToken();
  return token;
}

async function dbOps(): Promise<mysql.QueryResult | undefined> {
  try {
    // Obtain auth token
    const token = await createAuthToken();
```

```
const conn = await mysql.createConnection({
  host: proxy_host_name,
  user: db_user_name,
  password: token,
  database: db_name,
  ssl: 'Amazon RDS' // Ensure you have the CA bundle for SSL connection
});
const [rows, fields] = await conn.execute('SELECT ? + ? AS sum', [3, 2]);
console.log('result:', rows);
return rows;
}
catch (err) {
  console.log(err);
}
}

export const lambdaHandler = async (event: any): Promise<{ statusCode: number; body:
string }> => {
  // Execute database flow
  const result = await dbOps();

  // Return error is result is undefined
  if (result == undefined)
    return {
      statusCode: 500,
      body: JSON.stringify(`Error with connection to DB host`)
    }

  // Return result
  return {
    statusCode: 200,
    body: JSON.stringify(`The selected sum is: ${result[0].sum}`)
  };
};
```

Beispiele für Amazon RDS Data Service mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie AWS SDK für JavaScript (v3) mit Amazon RDS Data Service verwenden.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb eines Service aufrufen oder mit anderen AWS-Services kombinieren.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kodex finden.

Themen

- [Szenarien](#)

Szenarien

Erstellen eines Trackers für Aurora-Serverless-Arbeitsaufgaben

Das folgende Codebeispiel zeigt, wie eine Webanwendung erstellt wird, die Arbeitselemente in einer Datenbank von Amazon Aurora Serverless verfolgt und mithilfe von Amazon Simple Email Service (Amazon SES) Berichte sendet.

SDK für JavaScript (v3)

Zeigt, wie Sie mit AWS SDK für JavaScript (v3) eine Webanwendung erstellen, die Arbeitselemente in einer Amazon Aurora Datenbank verfolgt und Berichte mithilfe von Amazon Simple Email Service (Amazon SES) per E-Mail versendet. In diesem Beispiel wird ein mit React.js erstelltes Frontend verwendet, um mit einem Express-Node.js-Backend zu interagieren.

- Integrieren Sie eine React.js Webanwendung mit AWS-Services.
- Auflisten, hinzufügen und aktualisieren von Elementen in einer Aurora-Tabelle.
- Senden Sie einen E-Mail-Bericht über gefilterte Arbeitselemente mit Amazon SES.
- Stellen Sie Beispielressourcen mit dem mitgelieferten AWS CloudFormation Skript bereit und verwalten Sie sie.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

In diesem Beispiel verwendete Dienste

- Aurora
- Amazon RDS

- Amazon RDS Data Service
- Amazon SES

Amazon Redshift Redshift-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK für JavaScript (v3) mit Amazon Redshift Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kodex finden.

Themen

- [Aktionen](#)

Aktionen

CreateCluster

Das folgende Codebeispiel zeigt, wie Sie `CreateCluster`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie den Client.

```
import { RedshiftClient } from "@aws-sdk/client-redshift";
// Set the AWS Region.
const REGION = "REGION";
```

```
//Set the Redshift Service Object
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

Erstellen Sie den -Cluster.

```
// Import required AWS SDK clients and commands for Node.js
import { CreateClusterCommand } from "@aws-sdk/client-redshift";
import { redshiftClient } from "../libs/redshiftClient.js";

const params = {
  ClusterIdentifier: "CLUSTER_NAME", // Required
  NodeType: "NODE_TYPE", //Required
  MasterUsername: "MASTER_USER_NAME", // Required - must be lowercase
  MasterUserPassword: "MASTER_USER_PASSWORD", // Required - must contain at least
  one uppercase letter, and one number
  ClusterType: "CLUSTER_TYPE", // Required
  IAMRoleARN: "IAM_ROLE_ARN", // Optional - the ARN of an IAM role with permissions
  your cluster needs to access other AWS services on your behalf, such as Amazon S3.
  ClusterSubnetGroupName: "CLUSTER_SUBNET_GROUPNAME", //Optional - the name of a
  cluster subnet group to be associated with this cluster. Defaults to 'default' if
  not specified.
  DBName: "DATABASE_NAME", // Optional - defaults to 'dev' if not specified
  Port: "PORT_NUMBER", // Optional - defaults to '5439' if not specified
};

const run = async () => {
  try {
    const data = await redshiftClient.send(new CreateClusterCommand(params));
    console.log(
      `Cluster ${data.Cluster.ClusterIdentifier} successfully created`,
    );
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};

run();
```

- Einzelheiten zur API finden Sie [CreateCluster](#) in der AWS SDK für JavaScript API-Referenz.

DeleteCluster

Das folgende Codebeispiel zeigt die Verwendung `DeleteCluster`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie den Client.

```
import { RedshiftClient } from "@aws-sdk/client-redshift";
// Set the AWS Region.
const REGION = "REGION";
//Set the Redshift Service Object
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

Erstellen Sie den -Cluster.

```
// Import required AWS SDK clients and commands for Node.js
import { DeleteClusterCommand } from "@aws-sdk/client-redshift";
import { redshiftClient } from "../libs/redshiftClient.js";

const params = {
  ClusterIdentifier: "CLUSTER_NAME",
  SkipFinalClusterSnapshot: false,
  FinalClusterSnapshotIdentifier: "CLUSTER_SNAPSHOT_ID",
};

const run = async () => {
  try {
    const data = await redshiftClient.send(new DeleteClusterCommand(params));
    console.log("Success, cluster deleted. ", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
}
```

```
};  
run();
```

- Einzelheiten zur API finden Sie [DeleteCluster](#) in der AWS SDK für JavaScript API-Referenz.

DescribeClusters

Das folgende Codebeispiel zeigt die Verwendung `DescribeClusters`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie den Client.

```
import { RedshiftClient } from "@aws-sdk/client-redshift";  
// Set the AWS Region.  
const REGION = "REGION";  
//Set the Redshift Service Object  
const redshiftClient = new RedshiftClient({ region: REGION });  
export { redshiftClient };
```

Beschreiben Sie Ihre Cluster.

```
// Import required AWS SDK clients and commands for Node.js  
import { DescribeClustersCommand } from "@aws-sdk/client-redshift";  
import { redshiftClient } from "../libs/redshiftClient.js";  
  
const params = {  
  ClusterIdentifier: "CLUSTER_NAME",  
};  
  
const run = async () => {  
  try {  
    const data = await redshiftClient.send(new DescribeClustersCommand(params));
```

```
    console.log("Success", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
run();
```

- Einzelheiten zur API finden Sie [DescribeClusters](#) in der AWS SDK für JavaScript API-Referenz.

ModifyCluster

Das folgende Codebeispiel zeigt die Verwendung `ModifyCluster`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie den Client.

```
import { RedshiftClient } from "@aws-sdk/client-redshift";
// Set the AWS Region.
const REGION = "REGION";
//Set the Redshift Service Object
const redshiftClient = new RedshiftClient({ region: REGION });
export { redshiftClient };
```

Ändern Sie einen Cluster.

```
// Import required AWS SDK clients and commands for Node.js
import { ModifyClusterCommand } from "@aws-sdk/client-redshift";
import { redshiftClient } from "../libs/redshiftClient.js";

// Set the parameters
const params = {
```

```
ClusterIdentifier: "CLUSTER_NAME",
MasterUserPassword: "NEW_MASTER_USER_PASSWORD",
};

const run = async () => {
  try {
    const data = await redshiftClient.send(new ModifyClusterCommand(params));
    console.log("Success was modified.", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};

run();
```

- Einzelheiten zur API finden Sie [ModifyCluster](#) in der AWS SDK für JavaScript API-Referenz.

Amazon Rekognition Rekognition-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK für JavaScript (v3) mit Amazon Rekognition Aktionen ausführen und allgemeine Szenarien implementieren.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb eines Service aufrufen oder mit anderen AWS-Services kombinieren.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kodex finden.

Themen

- [Szenarien](#)

Szenarien

Erstellen einer Serverless-Anwendung zur Verwaltung von Fotos

Das folgende Codebeispiel zeigt, wie eine Serverless-Anwendung erstellt wird, mit der Benutzer Fotos mithilfe von Labels erstellen können.

SDK für JavaScript (v3)

Zeigt, wie eine Anwendung zur Verwaltung von Fotobeständen entwickelt wird, die mithilfe von Amazon Rekognition Labels in Bildern erkennt und sie für einen späteren Abruf speichert.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

Einen tiefen Einblick in den Ursprung dieses Beispiels finden Sie im Beitrag in der [AWS - Community](#).

In diesem Beispiel verwendete Dienste

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Erkennen von Objekten in Bildern

Die folgenden Codebeispiele zeigen, wie man eine App erstellt, die Amazon Rekognition verwendet, um Objekte nach Kategorien in Bildern zu erkennen.

SDK für JavaScript (v3)

Zeigt, wie Amazon Rekognition zusammen mit dem verwendet wird, um eine App AWS SDK für JavaScript zu erstellen, die Amazon Rekognition verwendet, um Objekte nach Kategorien in Bildern zu identifizieren, die sich in einem Amazon Simple Storage Service (Amazon S3) -Bucket befinden. Die App sendet dem Administrator eine E-Mail-Benachrichtigung mit den Ergebnissen über Amazon Simple Email Service (Amazon SES).

Lernen Sie Folgendes:

- Erstellen Sie mit Amazon Cognito einen nicht authentifizierten Benutzer.
- Analysieren Sie mit Amazon Rekognition Bilder für Objekte.
- Verifizieren Sie eine E-Mail-Adresse für Amazon SES.
- Senden Sie eine E-Mail-Benachrichtigung mit Amazon SES.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter. [GitHub](#)

In diesem Beispiel verwendete Dienste

- Amazon Rekognition
- Amazon S3
- Amazon SES

Amazon S3 S3-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK für JavaScript (v3) mit Amazon S3 Aktionen ausführen und allgemeine Szenarien implementieren.

Bei Grundlagen handelt es sich um Codebeispiele, die Ihnen zeigen, wie Sie die wesentlichen Vorgänge innerhalb eines Services ausführen.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien anzeigen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb eines Service aufrufen oder mit anderen AWS-Services kombinieren.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kodex finden.

Themen

- [Erste Schritte](#)
- [Grundlagen](#)
- [Aktionen](#)
- [Szenarien](#)
- [Serverless-Beispiele](#)

Erste Schritte

Hello Amazon S3

Das folgende Codebeispiel zeigt, wie Sie mit Amazon S3 beginnen können.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import {
  paginateListBuckets,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * List the S3 buckets in your configured AWS account.
 */
export const helloS3 = async () => {
  // When no region or credentials are provided, the SDK will use the
  // region and credentials from the local AWS config.
  const client = new S3Client({});

  try {
    /**
     * @type { import("@aws-sdk/client-s3").Bucket[] }
     */
    const buckets = [];

    for await (const page of paginateListBuckets({ client }, {})) {
      buckets.push(...page.Buckets);
    }
    console.log("Buckets: ");
    console.log(buckets.map((bucket) => bucket.Name).join("\n"));
    return buckets;
  } catch (caught) {
    // ListBuckets does not throw any modeled errors. Any error caught
```

```
// here will be something generic like `AccessDenied`.
if (caught instanceof S3ServiceException) {
  console.error(`${caught.name}: ${caught.message}`);
} else {
  // Something besides S3 failed.
  throw caught;
}
};
```

- Einzelheiten zur API finden Sie [ListBuckets](#) in der AWS SDK für JavaScript API-Referenz.

Grundlagen

Kennenlernen der Grundlagen

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Erstellen Sie einen Bucket und laden Sie eine Datei in ihn hoch.
- Laden Sie ein Objekt aus einem Bucket herunter.
- Kopieren Sie ein Objekt in einen Unterordner eines Buckets.
- Listen Sie die Objekte in einem Bucket auf.
- Löschen Sie die Bucket-Objekte und den Bucket.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Importieren Sie zunächst alle erforderlichen Module.

```
// Used to check if currently running file is this file.
import { fileURLToPath } from "node:url";
import { readdirSync, readFileSync, writeFileSync } from "node:fs";
```

```
// Local helper utils.
import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import { Prompter } from "@aws-doc-sdk-examples/lib/prompter.js";
import { wrapText } from "@aws-doc-sdk-examples/lib/utils/util-string.js";

import {
  S3Client,
  CreateBucketCommand,
  PutObjectCommand,
  ListObjectsCommand,
  CopyObjectCommand,
  GetObjectCommand,
  DeleteObjectsCommand,
  DeleteBucketCommand,
} from "@aws-sdk/client-s3";
```

Die vorherigen Importe verweisen auf einige hilfreiche Dienstprogramme. Diese Dienstprogramme befinden sich lokal in dem GitHub Repository, auf das am Anfang dieses Abschnitts verwiesen wird. Zu Ihrer Information finden Sie im Folgenden Implementierungen dieser Dienstprogramme.

```
export const dirnameFromMetaUrl = (metaUrl) =>
  fileURLToPath(new URL(".", metaUrl));

import { select, input, confirm, checkbox, password } from "@inquirer/prompts";

export class Prompter {
  /**
   * @param {{ message: string, choices: { name: string, value: string }[] }} options
   */
  select(options) {
    return select(options);
  }

  /**
   * @param {{ message: string }} options
   */
  input(options) {
    return input(options);
  }

  /**
   * @param {{ message: string }} options
```

```

    */
    password(options) {
        return password({ ...options, mask: true });
    }

    /**
     * @param {string} prompt
     */
    checkContinue = async (prompt = "") => {
        const prefix = prompt && `${prompt} `;
        const ok = await this.confirm({
            message: `${prefix}Continue?`,
        });
        if (!ok) throw new Error("Exiting...");
    };

    /**
     * @param {{ message: string }} options
     */
    confirm(options) {
        return confirm(options);
    }

    /**
     * @param {{ message: string, choices: { name: string, value: string }[] }} options
     */
    checkbox(options) {
        return checkbox(options);
    }
}

export const wrapText = (text, char = "=") => {
    const rule = char.repeat(80);
    return `${rule}\n    ${text}\n${rule}\n`;
};

```

Objekte werden in „Buckets“ gespeichert. Definieren wir eine Funktion zum Erstellen eines neuen Buckets.

```

export const createBucket = async () => {
    const bucketName = await prompter.input({
        message: "Enter a bucket name. Bucket names must be globally unique:",
    });
};

```

```
});  
const command = new CreateBucketCommand({ Bucket: bucketName });  
await s3Client.send(command);  
console.log("Bucket created successfully.\n");  
return bucketName;  
};
```

Buckets enthalten „Objekte“. Diese Funktion lädt den Inhalt eines Verzeichnisses als Objekte in Ihren Bucket hoch.

```
export const uploadFilesToBucket = async ({ bucketName, folderPath }) => {  
  console.log(`Uploading files from ${folderPath}\n`);  
  const keys = readdirSync(folderPath);  
  const files = keys.map((key) => {  
    const filePath = `${folderPath}/${key}`;  
    const fileContent = readFileSync(filePath);  
    return {  
      Key: key,  
      Body: fileContent,  
    };  
  });  
  
  for (const file of files) {  
    await s3Client.send(  
      new PutObjectCommand({  
        Bucket: bucketName,  
        Body: file.Body,  
        Key: file.Key,  
      })  
    );  
    console.log(`${file.Key} uploaded successfully.`);  
  }  
};
```

Überprüfen Sie nach dem Hochladen von Objekten, ob sie korrekt hochgeladen wurden. Sie können dafür verwenden `ListObjects`. Sie werden die Eigenschaft „Key“ verwenden, die Antwort enthält jedoch noch weitere nützliche Eigenschaften.

```
export const listFilesInBucket = async ({ bucketName }) => {  
  const command = new ListObjectsCommand({ Bucket: bucketName });
```

```
const { Contents } = await s3Client.send(command);
const contentsList = Contents.map((c) => ` • ${c.Key}`).join("\n");
console.log("\nHere's a list of files in the bucket:");
console.log(`${contentsList}\n`);
};
```

Gelegentlich möchten Sie vielleicht ein Objekt von einem Bucket in einen anderen kopieren. Verwenden Sie dafür den CopyObject Befehl.

```
export const copyFileFromBucket = async ({ destinationBucket }) => {
  const proceed = await prompter.confirm({
    message: "Would you like to copy an object from another bucket?",
  });

  if (!proceed) {
    return;
  }
  const copy = async () => {
    try {
      const sourceBucket = await prompter.input({
        message: "Enter source bucket name:",
      });
      const sourceKey = await prompter.input({
        message: "Enter source key:",
      });
      const destinationKey = await prompter.input({
        message: "Enter destination key:",
      });

      const command = new CopyObjectCommand({
        Bucket: destinationBucket,
        CopySource: `${sourceBucket}/${sourceKey}`,
        Key: destinationKey,
      });
      await s3Client.send(command);
      await copyFileFromBucket({ destinationBucket });
    } catch (err) {
      console.error("Copy error.");
      console.error(err);
      const retryAnswer = await prompter.confirm({ message: "Try again?" });
      if (retryAnswer) {
        await copy();
      }
    }
  };
};
```

```
    }  
  }  
};  
await copy();  
};
```

Es gibt keine SDK-Methode zum Abrufen mehrerer Objekte aus einem Bucket. Stattdessen erstellen Sie eine Liste von Objekten, die Sie herunterladen und durchlaufen können.

```
export const downloadFilesFromBucket = async ({ bucketName }) => {  
  const { Contents } = await s3Client.send(  
    new ListObjectsCommand({ Bucket: bucketName }),  
  );  
  const path = await prompter.input({  
    message: "Enter destination path for files:",  
  });  
  
  for (const content of Contents) {  
    const obj = await s3Client.send(  
      new GetObjectCommand({ Bucket: bucketName, Key: content.Key }),  
    );  
    writeFileSync(  
      `${path}/${content.Key}`,  
      await obj.Body.transformToByteArray(),  
    );  
  }  
  console.log("Files downloaded successfully.\n");  
};
```

Es ist Zeit, Ihre Ressourcen zu bereinigen. Ein Bucket muss leer sein, bevor er gelöscht werden kann. Mit diesen beiden Funktionen leeren und löschen Sie den Bucket.

```
export const emptyBucket = async ({ bucketName }) => {  
  const listObjectsCommand = new ListObjectsCommand({ Bucket: bucketName });  
  const { Contents } = await s3Client.send(listObjectsCommand);  
  const keys = Contents.map((c) => c.Key);  
  
  const deleteObjectsCommand = new DeleteObjectsCommand({  
    Bucket: bucketName,  
    Delete: { Objects: keys.map((key) => ({ Key: key })) },  
  });
```

```
});  
await s3Client.send(deleteObjectsCommand);  
console.log(`${bucketName} emptied successfully.\n`);  
};  
  
export const deleteBucket = async ({ bucketName }) => {  
  const command = new DeleteBucketCommand({ Bucket: bucketName });  
  await s3Client.send(command);  
  console.log(`${bucketName} deleted successfully.\n`);  
};
```

Die Funktion „main“ fasst alles zusammen. Wenn Sie diese Datei direkt ausführen, wird die Funktion „main“ aufgerufen.

```
const main = async () => {  
  const OBJECT_DIRECTORY = `${dirnameFromMetaUrl(  
    import.meta.url,  
  )}../../../../../resources/sample_files/.sample_media`;  
  
  try {  
    console.log(wrapText("Welcome to the Amazon S3 getting started example."));  
    console.log("Let's create a bucket.");  
    const bucketName = await createBucket();  
    await prompter.confirm({ message: continueMessage });  
  
    console.log(wrapText("File upload."));  
    console.log(  
      "I have some default files ready to go. You can edit the source code to  
provide your own.",  
    );  
    await uploadFilesToBucket({  
      bucketName,  
      folderPath: OBJECT_DIRECTORY,  
    });  
  
    await listFilesInBucket({ bucketName });  
    await prompter.confirm({ message: continueMessage });  
  
    console.log(wrapText("Copy files."));  
    await copyFileFromBucket({ destinationBucket: bucketName });  
    await listFilesInBucket({ bucketName });  
    await prompter.confirm({ message: continueMessage });
```



```
console.log(wrapText("Download files.));  
await downloadFilesFromBucket({ bucketName });  
  
console.log(wrapText("Clean up.));  
await emptyBucket({ bucketName });  
await deleteBucket({ bucketName });  
} catch (err) {  
  console.error(err);  
}  
};
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für JavaScript - API-Referenz.
 - [CopyObject](#)
 - [CreateBucket](#)
 - [DeleteBucket](#)
 - [DeleteObjects](#)
 - [GetObject](#)
 - [ListObjectsV2](#)
 - [PutObject](#)

Aktionen

CopyObject

Das folgende Codebeispiel zeigt, wie man es benutztCopyObject.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Kopieren Sie das Objekt.

```
import {
  S3Client,
  CopyObjectCommand,
  ObjectNotInActiveTierError,
  waitUntilObjectExists,
} from "@aws-sdk/client-s3";

/**
 * Copy an S3 object from one bucket to another.
 *
 * @param {{
 *   sourceBucket: string,
 *   sourceKey: string,
 *   destinationBucket: string,
 *   destinationKey: string }} config
 */
export const main = async ({
  sourceBucket,
  sourceKey,
  destinationBucket,
  destinationKey,
}) => {
  const client = new S3Client({});

  try {
    await client.send(
      new CopyObjectCommand({
        CopySource: `${sourceBucket}/${sourceKey}`,
        Bucket: destinationBucket,
        Key: destinationKey,
      }),
    );
    await waitUntilObjectExists(
      { client },
      { Bucket: destinationBucket, Key: destinationKey },
    );
    console.log(
      `Successfully copied ${sourceBucket}/${sourceKey} to ${destinationBucket}/${destinationKey}`,
    );
  } catch (caught) {
    if (caught instanceof ObjectNotInActiveTierError) {
      console.error(
```

```

        `Could not copy ${sourceKey} from ${sourceBucket}. Object is not in the
        active tier.`
    );
    } else {
        throw caught;
    }
}
};

```

Kopiert das Objekt unter der Bedingung, ETag dass es nicht mit dem angegebenen übereinstimmt.

```

import {
    CopyObjectCommand,
    NoSuchKey,
    S3Client,
    S3ServiceException,
} from "@aws-sdk/client-s3";

// Optionally edit the default key name of the copied object in 'object_name.json'
import data from "../scenarios/conditional-requests/object_name.json" assert {
    type: "json",
};

/**
 * Get a single object from a specified S3 bucket.
 * @param {{ sourceBucketName: string, sourceKeyName: string, destinationBucketName:
 * string, eTag: string }}
 */
export const main = async ({
    sourceBucketName,
    sourceKeyName,
    destinationBucketName,
    eTag,
}) => {
    const client = new S3Client({});
    const name = data.name;
    try {
        const response = await client.send(
            new CopyObjectCommand({
                CopySource: `${sourceBucketName}/${sourceKeyName}`,

```

```

        Bucket: destinationBucketName,
        Key: `${name}${sourceKeyName}`,
        CopySourceIfMatch: eTag,
    )),
    );
    console.log("Successfully copied object to bucket.");
} catch (caught) {
    if (caught instanceof NoSuchKey) {
        console.error(
            `Error from S3 while copying object "${sourceKeyName}" from
"${sourceBucketName}". No such key exists.`,
        );
    } else if (caught instanceof S3ServiceException) {
        console.error(
            `Unable to copy object "${sourceKeyName}" to bucket "${sourceBucketName}":
${caught.name}: ${caught.message}`,
        );
    } else {
        throw caught;
    }
}
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
    isMain,
    validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
    const options = {
        sourceBucketName: {
            type: "string",
            required: true,
        },
        sourceKeyName: {
            type: "string",
            required: true,
        },
        destinationBucketName: {
            type: "string",
            required: true,
        },
    },

```

```

    eTag: {
      type: "string",
      required: true,
    },
  };
  const results = parseArgs({ options });
  const { errors } = validateArgs({ options }, results);
  return { errors, results };
};

if (isMain(import.meta.url)) {
  const { errors, results } = loadArgs();
  if (!errors) {
    main(results.values);
  } else {
    console.error(errors.join("\n"));
  }
}

```

Kopieren Sie das Objekt unter der Bedingung, ETag dass es nicht mit dem angegebenen übereinstimmt.

```

import {
  CopyObjectCommand,
  NoSuchKey,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

// Optionally edit the default key name of the copied object in 'object_name.json'
import data from "../scenarios/conditional-requests/object_name.json" assert {
  type: "json",
};

/**
 * Get a single object from a specified S3 bucket.
 * @param {{ sourceBucketName: string, sourceKeyName: string, destinationBucketName:
  string, eTag: string }}
 */
export const main = async ({
  sourceBucketName,

```

```
    sourceKeyName,
    destinationBucketName,
    eTag,
  }) => {
    const client = new S3Client({});
    const name = data.name;

    try {
      const response = await client.send(
        new CopyObjectCommand({
          CopySource: `${sourceBucketName}/${sourceKeyName}`,
          Bucket: destinationBucketName,
          Key: `${name}${sourceKeyName}`,
          CopySourceIfNoneMatch: eTag,
        }),
      );
      console.log("Successfully copied object to bucket.");
    } catch (caught) {
      if (caught instanceof NoSuchKey) {
        console.error(
          `Error from S3 while copying object "${sourceKeyName}" from "${sourceBucketName}". No such key exists.`,
        );
      } else if (caught instanceof S3ServiceException) {
        console.error(
          `Unable to copy object "${sourceKeyName}" to bucket "${sourceBucketName}": ${caught.name}: ${caught.message}`,
        );
      } else {
        throw caught;
      }
    }
  };

// Call function if run directly
import { parseArgs } from "node:util";
import {
  isMain,
  validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
  const options = {
    sourceBucketName: {
```

```

        type: "string",
        required: true,
    },
    sourceKeyName: {
        type: "string",
        required: true,
    },
    destinationBucketName: {
        type: "string",
        required: true,
    },
    eTag: {
        type: "string",
        required: true,
    },
};
const results = parseArgs({ options });
const { errors } = validateArgs({ options }, results);
return { errors, results };
};

if (isMain(import.meta.url)) {
    const { errors, results } = loadArgs();
    if (!errors) {
        main(results.values);
    } else {
        console.error(errors.join("\n"));
    }
}

```

Kopieren Sie das Objekt unter der Bedingung, dass es in einem bestimmten Zeitraum erstellt oder geändert wurde.

```

import {
    CopyObjectCommand,
    NoSuchKey,
    S3Client,
    S3ServiceException,
} from "@aws-sdk/client-s3";

// Optionally edit the default key name of the copied object in 'object_name.json'

```

```
import data from "../scenarios/conditional-requests/object_name.json" assert {
  type: "json",
};

/**
 * Get a single object from a specified S3 bucket.
 * @param {{ sourceBucketName: string, sourceKeyName: string, destinationBucketName:
  string }}
 */
export const main = async ({
  sourceBucketName,
  sourceKeyName,
  destinationBucketName,
}) => {
  const date = new Date();
  date.setDate(date.getDate() - 1);

  const name = data.name;
  const client = new S3Client({});
  const copySource = `${sourceBucketName}/${sourceKeyName}`;
  const copiedKey = name + sourceKeyName;

  try {
    const response = await client.send(
      new CopyObjectCommand({
        CopySource: copySource,
        Bucket: destinationBucketName,
        Key: copiedKey,
        CopySourceIfModifiedSince: date,
      }),
    );
    console.log("Successfully copied object to bucket.");
  } catch (caught) {
    if (caught instanceof NoSuchKey) {
      console.error(
        `Error from S3 while copying object "${sourceKeyName}" from
        "${sourceBucketName}". No such key exists.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while copying object from ${sourceBucketName}.
        ${caught.name}: ${caught.message}`,
      );
    } else {

```



```
        throw caught;
    }
}
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
    isMain,
    validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
    const options = {
        sourceBucketName: {
            type: "string",
            required: true,
        },
        sourceKeyName: {
            type: "string",
            required: true,
        },
        destinationBucketName: {
            type: "string",
            required: true,
        },
    };
};

const results = parseArgs({ options });
const { errors } = validateArgs({ options }, results);
return { errors, results };
};

if (isMain(import.meta.url)) {
    const { errors, results } = loadArgs();
    if (!errors) {
        main(results.values);
    } else {
        console.error(errors.join("\n"));
    }
}
```

Kopiert das Objekt unter der Bedingung, dass es in einem bestimmten Zeitraum nicht erstellt oder geändert wurde.

```
import {
  CopyObjectCommand,
  NoSuchKey,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

// Optionally edit the default key name of the copied object in 'object_name.json'
import data from "../scenarios/conditional-requests/object_name.json" assert {
  type: "json",
};

/**
 * Get a single object from a specified S3 bucket.
 * @param {{ sourceBucketName: string, sourceKeyName: string, destinationBucketName:
  string }}
 */
export const main = async ({
  sourceBucketName,
  sourceKeyName,
  destinationBucketName,
}) => {
  const date = new Date();
  date.setDate(date.getDate() - 1);
  const client = new S3Client({});
  const name = data.name;
  const copiedKey = name + sourceKeyName;
  const copySource = `${sourceBucketName}/${sourceKeyName}`;

  try {
    const response = await client.send(
      new CopyObjectCommand({
        CopySource: copySource,
        Bucket: destinationBucketName,
        Key: copiedKey,
        CopySourceIfUnmodifiedSince: date,
      }),
    );
    console.log("Successfully copied object to bucket.");
  }
```

```
    } catch (caught) {
      if (caught instanceof NoSuchKey) {
        console.error(
          `Error from S3 while copying object "${sourceKeyName}" from
"${sourceBucketName}". No such key exists.`,
        );
      } else if (caught instanceof S3ServiceException) {
        console.error(
          `Error from S3 while copying object from ${sourceBucketName}.
${caught.name}: ${caught.message}`,
        );
      } else {
        throw caught;
      }
    }
  }
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
  isMain,
  validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
  const options = {
    sourceBucketName: {
      type: "string",
      required: true,
    },
    sourceKeyName: {
      type: "string",
      required: true,
    },
    destinationBucketName: {
      type: "string",
      required: true,
    },
  };
};

const results = parseArgs({ options });
const { errors } = validateArgs({ options }, results);
return { errors, results };
};
```

```
if (isMain(import.meta.url)) {
  const { errors, results } = loadArgs();
  if (!errors) {
    main(results.values);
  } else {
    console.error(errors.join("\n"));
  }
}
```

- Einzelheiten zur API finden Sie [CopyObject](#) in der AWS SDK für JavaScript API-Referenz.

CreateBucket

Das folgende Codebeispiel zeigt die Verwendung `CreateBucket`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie den Bucket.

```
import {
  BucketAlreadyExists,
  BucketAlreadyOwnedByYou,
  CreateBucketCommand,
  S3Client,
  waitUntilBucketExists,
} from "@aws-sdk/client-s3";

/**
 * Create an Amazon S3 bucket.
 * @param {{ bucketName: string }} config
 */
export const main = async ({ bucketName }) => {
  const client = new S3Client({});
```

```
try {
  const { Location } = await client.send(
    new CreateBucketCommand({
      // The name of the bucket. Bucket names are unique and have several other
      constraints.
      // See https://docs.aws.amazon.com/AmazonS3/latest/userguide/
      bucketnamingrules.html
      Bucket: bucketName,
    }),
  );
  await waitUntilBucketExists({ client }, { Bucket: bucketName });
  console.log(`Bucket created with location ${Location}`);
} catch (caught) {
  if (caught instanceof BucketAlreadyExists) {
    console.error(
      `The bucket "${bucketName}" already exists in another AWS account. Bucket
      names must be globally unique.`,
    );
  }
  // WARNING: If you try to create a bucket in the North Virginia region,
  // and you already own a bucket in that region with the same name, this
  // error will not be thrown. Instead, the call will return successfully
  // and the ACL on that bucket will be reset.
  else if (caught instanceof BucketAlreadyOwnedByYou) {
    console.error(
      `The bucket "${bucketName}" already exists in this AWS account.`,
    );
  } else {
    throw caught;
  }
}
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [CreateBucket](#) in der AWS SDK für JavaScript API-Referenz.

DeleteBucket

Das folgende Codebeispiel zeigt die Verwendung `DeleteBucket`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Löschen Sie den Bucket.

```
import {
  DeleteBucketCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Delete an Amazon S3 bucket.
 * @param {{ bucketName: string }}
 */
export const main = async ({ bucketName }) => {
  const client = new S3Client({});
  const command = new DeleteBucketCommand({
    Bucket: bucketName,
  });

  try {
    await client.send(command);
    console.log("Bucket was deleted.");
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while deleting bucket. The bucket doesn't exist.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while deleting the bucket. ${caught.name}:`,
        `${caught.message}`,
      );
    } else {

```

```
        throw caught;
    }
}
};
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [DeleteBucket](#) in der AWS SDK für JavaScript API-Referenz.

DeleteBucketPolicy

Das folgende Codebeispiel zeigt die Verwendung `DeleteBucketPolicy`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Löschen Sie die Bucket-Richtlinie.

```
import {
  DeleteBucketPolicyCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Remove the policy from an Amazon S3 bucket.
 * @param {{ bucketName: string }}
 */
export const main = async ({ bucketName }) => {
  const client = new S3Client({});

  try {
    await client.send(
      new DeleteBucketPolicyCommand({
        Bucket: bucketName,
      }),
    );
  }
```

```
    console.log(`Bucket policy deleted from "${bucketName}".`);
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while deleting policy from ${bucketName}. The bucket doesn't exist.`
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while deleting policy from ${bucketName}. ${caught.name}: ${caught.message}`
      );
    } else {
      throw caught;
    }
  }
};
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [DeleteBucketPolicy](#) in der AWS SDK für JavaScript API-Referenz.

DeleteBucketWebsite

Das folgende Codebeispiel zeigt die Verwendung `DeleteBucketWebsite`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Löschen Sie die Website-Konfiguration aus dem Bucket.

```
import {
```



```
    DeleteBucketWebsiteCommand,
    S3Client,
    S3ServiceException,
  } from "@aws-sdk/client-s3";

/**
 * Remove the website configuration for a bucket.
 * @param {{ bucketName: string }}
 */
export const main = async ({ bucketName }) => {
  const client = new S3Client({});

  try {
    await client.send(
      new DeleteBucketWebsiteCommand({
        Bucket: bucketName,
      }),
    );
    // The response code will be successful for both removed configurations and
    // configurations that did not exist in the first place.
    console.log(
      `The bucket "${bucketName}" is not longer configured as a website, or it never
was.`,
    );
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while removing website configuration from ${bucketName}. The
bucket doesn't exist.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while removing website configuration from ${bucketName}.
${caught.name}: ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [DeleteBucketWebsite](#) in der AWS SDK für JavaScript API-Referenz.

DeleteObject

Das folgende Codebeispiel zeigt die Verwendung `DeleteObject`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Löschen Sie ein Objekt.

```
import {
  DeleteObjectCommand,
  S3Client,
  S3ServiceException,
  waitUntilObjectNotExists,
} from "@aws-sdk/client-s3";

/**
 * Delete one object from an Amazon S3 bucket.
 * @param {{ bucketName: string, key: string }}
 */
export const main = async ({ bucketName, key }) => {
  const client = new S3Client({});

  try {
    await client.send(
      new DeleteObjectCommand({
        Bucket: bucketName,
        Key: key,
      }),
    );
  }
```

```
await waitUntilObjectNotExists(  
  { client },  
  { Bucket: bucketName, Key: key },  
);  
// A successful delete, or a delete for a non-existent object, both return  
// a 204 response code.  
console.log(  
  `The object "${key}" from bucket "${bucketName}" was deleted, or it didn't  
  exist.`,  
);  
} catch (caught) {  
  if (  
    caught instanceof S3ServiceException &&  
    caught.name === "NoSuchBucket"  
  ) {  
    console.error(  
      `Error from S3 while deleting object from ${bucketName}. The bucket doesn't  
      exist.`,  
    );  
  } else if (caught instanceof S3ServiceException) {  
    console.error(  
      `Error from S3 while deleting object from ${bucketName}. ${caught.name}:  
      ${caught.message}`,  
    );  
  } else {  
    throw caught;  
  }  
}  
};
```

- Einzelheiten zur API finden Sie [DeleteObject](#) in der AWS SDK für JavaScript API-Referenz.

DeleteObjects

Das folgende Codebeispiel zeigt die Verwendung `DeleteObjects`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Löschen Sie mehrere Objekte.

```
import {
  DeleteObjectsCommand,
  S3Client,
  S3ServiceException,
  waitUntilObjectNotExists,
} from "@aws-sdk/client-s3";

/**
 * Delete multiple objects from an S3 bucket.
 * @param {{ bucketName: string, keys: string[] }}
 */
export const main = async ({ bucketName, keys }) => {
  const client = new S3Client({});

  try {
    const { Deleted } = await client.send(
      new DeleteObjectsCommand({
        Bucket: bucketName,
        Delete: {
          Objects: keys.map((k) => ({ Key: k })),
        },
      }),
    );
    for (const key in keys) {
      await waitUntilObjectNotExists(
        { client },
        { Bucket: bucketName, Key: key },
      );
    }
    console.log(
      `Successfully deleted ${Deleted.length} objects from S3 bucket. Deleted objects:`,
    );
  }
};
```

```
    console.log(Deleted.map((d) => ` • ${d.Key}`).join("\n"));
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while deleting objects from ${bucketName}. The bucket doesn't
exist.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while deleting objects from ${bucketName}.  ${caught.name}:
${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Einzelheiten zur API finden Sie [DeleteObjects](#) in der AWS SDK für JavaScript API-Referenz.

GetBucketAcl

Das folgende Codebeispiel zeigt die Verwendung `GetBucketAcl`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Rufen Sie die ACL-Berechtigungen ab.

```
import {
  GetBucketAclCommand,
  S3Client,
```

```
S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Retrieves the Access Control List (ACL) for an S3 bucket.
 * @param {{ bucketName: string }}
 */
export const main = async ({ bucketName }) => {
  const client = new S3Client({});

  try {
    const response = await client.send(
      new GetBucketAclCommand({
        Bucket: bucketName,
      }),
    );
    console.log(`ACL for bucket "${bucketName}":`);
    console.log(JSON.stringify(response, null, 2));
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while getting ACL for ${bucketName}. The bucket doesn't exist.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while getting ACL for ${bucketName}. ${caught.name}: ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [GetBucketAcl](#) in der AWS SDK für JavaScript API-Referenz.

GetBucketCors

Das folgende Codebeispiel zeigt die Verwendung `GetBucketCors`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Rufen Sie die CORS-Richtlinie für den Bucket ab.

```
import {
  GetBucketCorsCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Log the Cross-Origin Resource Sharing (CORS) configuration information
 * set for the bucket.
 * @param {{ bucketName: string }}
 */
export const main = async ({ bucketName }) => {
  const client = new S3Client({});
  const command = new GetBucketCorsCommand({
    Bucket: bucketName,
  });

  try {
    const { CORSRules } = await client.send(command);
    console.log(JSON.stringify(CORSRules));
    CORSRules.forEach((cr, i) => {
      console.log(
        `\nCORSRule ${i + 1}`,
        `${"-".repeat(10)}`,
        `\nAllowedHeaders: ${cr.AllowedHeaders}`,
        `\nAllowedMethods: ${cr.AllowedMethods}`,
        `\nAllowedOrigins: ${cr.AllowedOrigins}`,
        `\nExposeHeaders: ${cr.ExposeHeaders}`,
        `\nMaxAgeSeconds: ${cr.MaxAgeSeconds}`,
      );
    });
  } catch (err) {
    if (err instanceof S3ServiceException) {
      console.log("S3ServiceException: ", err.message);
    } else {
      console.log("Unknown error: ", err);
    }
  }
};
```

```
    );
  });
} catch (caught) {
  if (
    caught instanceof S3ServiceException &&
    caught.name === "NoSuchBucket"
  ) {
    console.error(
      `Error from S3 while getting bucket CORS rules for ${bucketName}. The bucket
doesn't exist.`,
    );
  } else if (caught instanceof S3ServiceException) {
    console.error(
      `Error from S3 while getting bucket CORS rules for ${bucketName}.
${caught.name}: ${caught.message}`,
    );
  } else {
    throw caught;
  }
}
};
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [GetBucketCors](#) in der AWS SDK für JavaScript API-Referenz.

GetBucketPolicy

Das folgende Codebeispiel zeigt die Verwendung `GetBucketPolicy`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Rufen Sie die Bucket-Richtlinie ab.

```
import {
```



```

    GetBucketPolicyCommand,
    S3Client,
    S3ServiceException,
  } from "@aws-sdk/client-s3";

/**
 * Logs the policy for a specified bucket.
 * @param {{ bucketName: string }}
 */
export const main = async ({ bucketName }) => {
  const client = new S3Client({});

  try {
    const { Policy } = await client.send(
      new GetBucketPolicyCommand({
        Bucket: bucketName,
      }),
    );
    console.log(`Policy for "${bucketName}":\n${Policy}`);
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while getting policy from ${bucketName}. The bucket doesn't exist.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while getting policy from ${bucketName}. ${caught.name}: ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};

```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [GetBucketPolicy](#) in der AWS SDK für JavaScript API-Referenz.

GetBucketWebsite

Das folgende Codebeispiel zeigt die Verwendung `GetBucketWebsite`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Rufen Sie die Website-Konfiguration ab.

```
import {
  GetBucketWebsiteCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Log the website configuration for a bucket.
 * @param {{ bucketName }}
 */
export const main = async ({ bucketName }) => {
  const client = new S3Client({});

  try {
    const response = await client.send(
      new GetBucketWebsiteCommand({
        Bucket: bucketName,
      }),
    );
    console.log(
      `Your bucket is set up to host a website with the following configuration:\n
      ${JSON.stringify(response, null, 2)}`,
    );
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchWebsiteConfiguration"
    ) {
      console.error(
```

```

        `Error from S3 while getting website configuration for ${bucketName}. The
        bucket isn't configured as a website.` ,
    );
    } else if (caught instanceof S3ServiceException) {
        console.error(
            `Error from S3 while getting website configuration for ${bucketName}.
            ${caught.name}: ${caught.message}`,
        );
    } else {
        throw caught;
    }
}
};

```

- Einzelheiten zur API finden Sie [GetBucketWebsite](#) in der AWS SDK für JavaScript API-Referenz.

GetObject

Das folgende Codebeispiel zeigt die Verwendung `GetObject`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Laden Sie das Objekt herunter.

```

import {
    GetObjectCommand,
    NoSuchKey,
    S3Client,
    S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Get a single object from a specified S3 bucket.
 * @param {{ bucketName: string, key: string }}

```

```
*/
export const main = async ({ bucketName, key }) => {
  const client = new S3Client({});

  try {
    const response = await client.send(
      new GetObjectCommand({
        Bucket: bucketName,
        Key: key,
      }),
    );
    // The Body object also has 'transformToByteArray' and 'transformToWebStream'
    methods.
    const str = await response.Body.transformToString();
    console.log(str);
  } catch (caught) {
    if (caught instanceof NoSuchKey) {
      console.error(
        `Error from S3 while getting object "${key}" from "${bucketName}". No such
        key exists.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while getting object from ${bucketName}. ${caught.name}:
        ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};
```

Laden Sie das Objekt herunter, sofern es ETag mit dem bereitgestellten übereinstimmt.

```
import {
  GetObjectCommand,
  NoSuchKey,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";
```

```
/**
 * Get a single object from a specified S3 bucket.
 * @param {{ bucketName: string, key: string, eTag: string }}
 */
export const main = async ({ bucketName, key, eTag }) => {
  const client = new S3Client({});

  try {
    const response = await client.send(
      new GetObjectCommand({
        Bucket: bucketName,
        Key: key,
        IfMatch: eTag,
      }),
    );
    // The Body object also has 'transformToByteArray' and 'transformToWebStream'
    methods.
    const str = await response.Body.transformToString();
    console.log("Success. Here is text of the file:", str);
  } catch (caught) {
    if (caught instanceof NoSuchKey) {
      console.error(
        `Error from S3 while getting object "${key}" from "${bucketName}". No such
        key exists.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while getting object from ${bucketName}. ${caught.name}:
        ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
  isMain,
  validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
```

```

const options = {
  bucketName: {
    type: "string",
    required: true,
  },
  key: {
    type: "string",
    required: true,
  },
  eTag: {
    type: "string",
    required: true,
  },
};
const results = parseArgs({ options });
const { errors } = validateArgs({ options }, results);
return { errors, results };
};

if (isMain(import.meta.url)) {
  const { errors, results } = loadArgs();
  if (!errors) {
    main(results.values);
  } else {
    console.error(errors.join("\n"));
  }
}

```

Laden Sie das Objekt unter der Bedingung herunter, ETag dass es nicht mit dem angegebenen übereinstimmt.

```

import {
  GetObjectCommand,
  NoSuchKey,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Get a single object from a specified S3 bucket.
 * @param {{ bucketName: string, key: string, eTag: string }}

```

```
*/
export const main = async ({ bucketName, key, eTag }) => {
  const client = new S3Client({});

  try {
    const response = await client.send(
      new GetObjectCommand({
        Bucket: bucketName,
        Key: key,
        IfNoneMatch: eTag,
      }),
    );
    // The Body object also has 'transformToByteArray' and 'transformToWebStream'
    methods.
    const str = await response.Body.transformToString();
    console.log("Success. Here is text of the file:", str);
  } catch (caught) {
    if (caught instanceof NoSuchKey) {
      console.error(
        `Error from S3 while getting object "${key}" from "${bucketName}". No such
key exists.`
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while getting object from ${bucketName}. ${caught.name}:
${caught.message}`
      );
    } else {
      throw caught;
    }
  }
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
  isMain,
  validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
  const options = {
    bucketName: {
      type: "string",
```

```

        required: true,
    },
    key: {
        type: "string",
        required: true,
    },
    eTag: {
        type: "string",
        required: true,
    },
};
const results = parseArgs({ options });
const { errors } = validateArgs({ options }, results);
return { errors, results };
};

if (isMain(import.meta.url)) {
    const { errors, results } = loadArgs();
    if (!errors) {
        main(results.values);
    } else {
        console.error(errors.join("\n"));
    }
}

```

Laden das Objekt unter der Bedingung herunter, dass es in einem bestimmten Zeitraum erstellt oder geändert wurde.

```

import {
    GetObjectCommand,
    NoSuchKey,
    S3Client,
    S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Get a single object from a specified S3 bucket.
 * @param {{ bucketName: string, key: string }}
 */
export const main = async ({ bucketName, key }) => {
    const client = new S3Client({});

```



```
const date = new Date();
date.setDate(date.getDate() - 1);
try {
  const response = await client.send(
    new GetObjectCommand({
      Bucket: bucketName,
      Key: key,
      IfModifiedSince: date,
    }),
  );
  // The Body object also has 'transformToArray' and 'transformToWebStream'
  methods.
  const str = await response.Body.transformToString();
  console.log("Success. Here is text of the file:", str);
} catch (caught) {
  if (caught instanceof NoSuchKey) {
    console.error(
      `Error from S3 while getting object "${key}" from "${bucketName}". No such
key exists.`
    );
  } else if (caught instanceof S3ServiceException) {
    console.error(
      `Error from S3 while getting object from ${bucketName}. ${caught.name}:
${caught.message}`
    );
  } else {
    throw caught;
  }
}
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
  isMain,
  validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
  const options = {
    bucketName: {
      type: "string",
      required: true,
    },
  },
```

```

    key: {
      type: "string",
      required: true,
    },
  };
  const results = parseArgs({ options });
  const { errors } = validateArgs({ options }, results);
  return { errors, results };
};

if (isMain(import.meta.url)) {
  const { errors, results } = loadArgs();
  if (!errors) {
    main(results.values);
  } else {
    console.error(errors.join("\n"));
  }
}

```

Laden das Objekt unter der Bedingung herunter, dass es in einem bestimmten Zeitraum nicht erstellt oder geändert wurde.

```

import {
  GetObjectCommand,
  NoSuchKey,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Get a single object from a specified S3 bucket.
 * @param {{ bucketName: string, key: string }}
 */
export const main = async ({ bucketName, key }) => {
  const client = new S3Client({});
  const date = new Date();
  date.setDate(date.getDate() - 1);
  try {
    const response = await client.send(
      new GetObjectCommand({
        Bucket: bucketName,

```

```
        Key: key,
        IfUnmodifiedSince: date,
    })),
    );
    // The Body object also has 'transformToByteArray' and 'transformToWebStream'
    methods.
    const str = await response.Body.transformToString();
    console.log("Success. Here is text of the file:", str);
} catch (caught) {
    if (caught instanceof NoSuchKey) {
        console.error(
            `Error from S3 while getting object "${key}" from "${bucketName}". No such
            key exists.`
        );
    } else if (caught instanceof S3ServiceException) {
        console.error(
            `Error from S3 while getting object from ${bucketName}. ${caught.name}:
            ${caught.message}`
        );
    } else {
        throw caught;
    }
}
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
    isMain,
    validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
    const options = {
        bucketName: {
            type: "string",
            required: true,
        },
        key: {
            type: "string",
            required: true,
        },
    };
    const results = parseArgs({ options });
```

```

    const { errors } = validateArgs({ options }, results);
    return { errors, results };
};

if (isMain(import.meta.url)) {
    const { errors, results } = loadArgs();
    if (!errors) {
        main(results.values);
    } else {
        console.error(errors.join("\n"));
    }
}

```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [GetObject](#) in der AWS SDK für JavaScript API-Referenz.

GetObjectLegalHold

Das folgende Codebeispiel zeigt die Verwendung `GetObjectLegalHold`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```

import {
    GetObjectLegalHoldCommand,
    S3Client,
    S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Get an object's current legal hold status.
 * @param {{ bucketName: string, key: string }}
 */
export const main = async ({ bucketName, key }) => {

```

```
const client = new S3Client({});

try {
  const response = await client.send(
    new GetObjectLegalHoldCommand({
      Bucket: bucketName,
      Key: key,
      // Optionally, you can provide additional parameters
      // ExpectedBucketOwner: "<account ID that is expected to own the bucket>",
      // VersionId: "<the specific version id of the object to check>",
    }),
  );
  console.log(`Legal Hold Status: ${response.LegalHold.Status}`);
} catch (caught) {
  if (
    caught instanceof S3ServiceException &&
    caught.name === "NoSuchBucket"
  ) {
    console.error(
      `Error from S3 while getting legal hold status for ${key} in ${bucketName}.
The bucket doesn't exist.`,
    );
  } else if (caught instanceof S3ServiceException) {
    console.error(
      `Error from S3 while getting legal hold status for ${key} in ${bucketName}
from ${bucketName}.  ${caught.name}: ${caught.message}`,
    );
  } else {
    throw caught;
  }
}

// Call function if run directly
import { parseArgs } from "node:util";
import {
  isMain,
  validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
  const options = {
    bucketName: {
      type: "string",
```

```
        required: true,
      },
      key: {
        type: "string",
        required: true,
      },
    },
  };
  const results = parseArgs({ options });
  const { errors } = validateArgs({ options }, results);
  return { errors, results };
};

if (isMain(import.meta.url)) {
  const { errors, results } = loadArgs();
  if (!errors) {
    main(results.values);
  } else {
    console.error(errors.join("\n"));
  }
}
```

- Einzelheiten zur API finden Sie [GetObjectLegalHold](#) in der AWS SDK für JavaScript API-Referenz.

GetObjectLockConfiguration

Das folgende Codebeispiel zeigt die Verwendung `GetObjectLockConfiguration`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import {
  GetObjectLockConfigurationCommand,
  S3Client,
  S3ServiceException,
```

```

} from "@aws-sdk/client-s3";

/**
 * Gets the Object Lock configuration for a bucket.
 * @param {{ bucketName: string }}
 */
export const main = async ({ bucketName }) => {
  const client = new S3Client({});

  try {
    const { ObjectLockConfiguration } = await client.send(
      new GetObjectLockConfigurationCommand({
        Bucket: bucketName,
        // Optionally, you can provide additional parameters
        // ExpectedBucketOwner: "<account ID that is expected to own the bucket>",
      }),
    );
    console.log(
      `Object Lock Configuration:\n${JSON.stringify(ObjectLockConfiguration)}`,
    );
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while getting object lock configuration for ${bucketName}.
The bucket doesn't exist.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while getting object lock configuration for ${bucketName}.
${caught.name}: ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
  isMain,

```

```
    validateArgs,  
  } from "@aws-doc-sdk-examples/lib/utils/util-node.js";  
  
const loadArgs = () => {  
  const options = {  
    bucketName: {  
      type: "string",  
      required: true,  
    },  
  };  
  const results = parseArgs({ options });  
  const { errors } = validateArgs({ options }, results);  
  return { errors, results };  
};  
  
if (isMain(import.meta.url)) {  
  const { errors, results } = loadArgs();  
  if (!errors) {  
    main(results.values);  
  } else {  
    console.error(errors.join("\n"));  
  }  
}
```

- Einzelheiten zur API finden Sie [GetObjectLockConfiguration](#) in der AWS SDK für JavaScript API-Referenz.

GetObjectRetention

Das folgende Codebeispiel zeigt die Verwendung `GetObjectRetention`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.


```
import {
  GetObjectRetentionCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Log the "RetainUntilDate" for an object in an S3 bucket.
 * @param {{ bucketName: string, key: string }}
 */
export const main = async ({ bucketName, key }) => {
  const client = new S3Client({});

  try {
    const { Retention } = await client.send(
      new GetObjectRetentionCommand({
        Bucket: bucketName,
        Key: key,
      }),
    );
    console.log(
      `${key} in ${bucketName} will be retained until ${Retention.RetainUntilDate}`,
    );
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchObjectLockConfiguration"
    ) {
      console.warn(
        `The object "${key}" in the bucket "${bucketName}" does not have an ObjectLock configuration.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while getting object retention settings for "${bucketName}". ${caught.name}: ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};

// Call function if run directly
```

```
import { parseArgs } from "node:util";
import {
  isMain,
  validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
  const options = {
    bucketName: {
      type: "string",
      required: true,
    },
    key: {
      type: "string",
      required: true,
    },
  };
  const results = parseArgs({ options });
  const { errors } = validateArgs({ options }, results);
  return { errors, results };
};

if (isMain(import.meta.url)) {
  const { errors, results } = loadArgs();
  if (!errors) {
    main(results.values);
  } else {
    console.error(errors.join("\n"));
  }
}
```

- Einzelheiten zur API finden Sie [GetObjectRetention](#) in der AWS SDK für JavaScript API-Referenz.

ListBuckets

Das folgende Codebeispiel zeigt die Verwendung `ListBuckets`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Listen Sie die Buckets auf.

```
import {
  paginateListBuckets,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * List the Amazon S3 buckets in your account.
 */
export const main = async () => {
  const client = new S3Client({});
  /** @type {?import('@aws-sdk/client-s3').Owner} */
  let Owner = null;

  /** @type {import('@aws-sdk/client-s3').Bucket[]} */
  const Buckets = [];

  try {
    const paginator = paginateListBuckets({ client }, {});

    for await (const page of paginator) {
      if (!Owner) {
        Owner = page.Owner;
      }

      Buckets.push(...page.Buckets);
    }

    console.log(
      `${Owner.DisplayName} owns ${Buckets.length} bucket${
        Buckets.length === 1 ? "" : "s"
      }`,
    );
  }
```

```
    console.log(`${Buckets.map((b) => ` • ${b.Name}` ).join("\n")}`);
  } catch (caught) {
    if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while listing buckets.  ${caught.name}: ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [ListBuckets](#) in der AWS SDK für JavaScript API-Referenz.

ListObjectsV2

Das folgende Codebeispiel zeigt die Verwendung `ListObjectsV2`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Listen Sie alle Objekte in Ihrem Bucket auf. Wenn es mehr als ein Objekt gibt, `IsTruncated` `NextContinuationToken` wird es verwendet, um über die gesamte Liste zu iterieren.

```
import {
  S3Client,
  S3ServiceException,
  // This command supersedes the ListObjectsCommand and is the recommended way to
  list objects.
  paginateListObjectsV2,
} from "@aws-sdk/client-s3";

/**
 * Log all of the object keys in a bucket.
```

```

* @param {{ bucketName: string, pageSize: string }}
*/
export const main = async ({ bucketName, pageSize }) => {
  const client = new S3Client({});
  /** @type {string[][]} */
  const objects = [];
  try {
    const paginator = paginateListObjectsV2(
      { client, /* Max items per page */ pageSize: Number.parseInt(pageSize) },
      { Bucket: bucketName },
    );

    for await (const page of paginator) {
      objects.push(page.Contents.map((o) => o.Key));
    }
    objects.forEach((objectList, pageNum) => {
      console.log(
        `Page ${pageNum + 1}\n-----\n${objectList.map((o) => `•
${o}`)}.join("\n")\n`,
      );
    });
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while listing objects for "${bucketName}". The bucket doesn't
exist.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while listing objects for "${bucketName}". ${caught.name}:
${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};

```

- Einzelheiten zur API finden Sie unter [ListObjectsV2](#) in der AWS SDK für JavaScript API-Referenz.

PutBucketAcl

Das folgende Codebeispiel zeigt die Verwendung `PutBucketAcl`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Legen Sie die Bucket-ACL fest.

```
import {
  PutBucketAclCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Grant read access to a user using their canonical AWS account ID.
 *
 * Most Amazon S3 use cases don't require the use of access control lists (ACLs).
 * We recommend that you disable ACLs, except in unusual circumstances where
 * you need to control access for each object individually. Consider a policy
 * instead.
 * For more information see https://docs.aws.amazon.com/AmazonS3/latest/userguide/
 * bucket-policies.html.
 * @param {{ bucketName: string, granteeCanonicalUserId: string,
 * ownerCanonicalUserId }}
 */
export const main = async ({
  bucketName,
  granteeCanonicalUserId,
  ownerCanonicalUserId,
}) => {
  const client = new S3Client({});
  const command = new PutBucketAclCommand({
```

```

    Bucket: bucketName,
    AccessControlPolicy: {
      Grants: [
        {
          Grantee: {
            // The canonical ID of the user. This ID is an obfuscated form of your
            AWS account number.
            // It's unique to Amazon S3 and can't be found elsewhere.
            // For more information, see https://docs.aws.amazon.com/AmazonS3/
latest/userguide/finding-canonical-user-id.html.
            ID: granteeCanonicalUserId,
            Type: "CanonicalUser",
          },
          // One of FULL_CONTROL | READ | WRITE | READ_ACP | WRITE_ACP
          // https://docs.aws.amazon.com/AmazonS3/latest/API/
API_Grant.html#AmazonS3-Type-Grant-Permission
          Permission: "READ",
        },
      ],
      Owner: {
        ID: ownerCanonicalUserId,
      },
    },
  });

  try {
    await client.send(command);
    console.log(`Granted READ access to ${bucketName}`);
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while setting ACL for bucket ${bucketName}. The bucket
doesn't exist.`
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while setting ACL for bucket ${bucketName}. ${caught.name}:
${caught.message}`
      );
    } else {
      throw caught;
    }
  }
}

```

```

    }
  }
};

```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [PutBucketAcl](#) in der AWS SDK für JavaScript API-Referenz.

PutBucketCors

Das folgende Codebeispiel zeigt die Verwendung `PutBucketCors`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Fügen Sie eine CORS-Regel hinzu.

```

import {
  PutBucketCorsCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Allows cross-origin requests to an S3 bucket by setting the CORS configuration.
 * @param {{ bucketName: string }}
 */
export const main = async ({ bucketName }) => {
  const client = new S3Client({});

  try {
    await client.send(
      new PutBucketCorsCommand({
        Bucket: bucketName,
        CORSConfiguration: {
          CORSRules: [
            {

```



```

        // Allow all headers to be sent to this bucket.
        AllowedHeaders: ["*"],
        // Allow only GET and PUT methods to be sent to this bucket.
        AllowedMethods: ["GET", "PUT"],
        // Allow only requests from the specified origin.
        AllowedOrigins: ["https://www.example.com"],
        // Allow the entity tag (ETag) header to be returned in the response.
The ETag header
        // The entity tag represents a specific version of the object. The
ETag reflects
        // changes only to the contents of an object, not its metadata.
        ExposeHeaders: ["ETag"],
        // How long the requesting browser should cache the preflight
response. After
        // this time, the preflight request will have to be made again.
        MaxAgeSeconds: 3600,
    },
],
},
)),
);
console.log(`Successfully set CORS rules for bucket: ${bucketName}`);
} catch (caught) {
    if (
        caught instanceof S3ServiceException &&
        caught.name === "NoSuchBucket"
    ) {
        console.error(
            `Error from S3 while setting CORS rules for ${bucketName}. The bucket
doesn't exist.`,
        );
    } else if (caught instanceof S3ServiceException) {
        console.error(
            `Error from S3 while setting CORS rules for ${bucketName}. ${caught.name}:
${caught.message}`,
        );
    } else {
        throw caught;
    }
}
};

```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).

- Einzelheiten zur API finden Sie [PutBucketCors](#) in der AWS SDK für JavaScript API-Referenz.

PutBucketPolicy

Das folgende Codebeispiel zeigt die Verwendung `PutBucketPolicy`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Fügen Sie die Richtlinie hinzu.

```
import {
  PutBucketPolicyCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Grant an IAM role GetObject access to all of the objects
 * in the provided bucket.
 * @param {{ bucketName: string, iamRoleArn: string }}
 */
export const main = async ({ bucketName, iamRoleArn }) => {
  const client = new S3Client({});
  const command = new PutBucketPolicyCommand({
    // This is a resource-based policy. For more information on resource-based
    policies,
    // see https://docs.aws.amazon.com/IAM/latest/UserGuide/
    access_policies.html#policies_resource-based.
    Policy: JSON.stringify({
      Version: "2012-10-17",
      Statement: [
        {
          Effect: "Allow",
          Principal: {
            AWS: iamRoleArn,
          },
        },
      ],
    }),
  });
```

```

        Action: "s3:GetObject",
        Resource: `arn:aws:s3:::${bucketName}/*`,
    },
],
)),
// Apply the preceding policy to this bucket.
Bucket: bucketName,
});

try {
    await client.send(command);
    console.log(
        `GetObject access to the bucket "${bucketName}" was granted to the provided
IAM role.`,
    );
} catch (caught) {
    if (
        caught instanceof S3ServiceException &&
        caught.name === "MalformedPolicy"
    ) {
        console.error(
            `Error from S3 while setting the bucket policy for the bucket
"${bucketName}". The policy was malformed.`,
        );
    } else if (caught instanceof S3ServiceException) {
        console.error(
            `Error from S3 while setting the bucket policy for the bucket
"${bucketName}". ${caught.name}: ${caught.message}`,
        );
    } else {
        throw caught;
    }
}
};

```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [PutBucketPolicy](#) in der AWS SDK für JavaScript API-Referenz.

PutBucketWebsite

Das folgende Codebeispiel zeigt die Verwendung `PutBucketWebsite`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Legen Sie die Website-Konfiguration fest.

```
import {
  PutBucketWebsiteCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Configure an Amazon S3 bucket to serve a static website.
 * Website access must also be granted separately. For more information
 * on setting the permissions for website access, see
 * https://docs.aws.amazon.com/AmazonS3/latest/userguide/
 * WebsiteAccessPermissionsReqd.html.
 */
@param {{ bucketName: string }}
export const main = async ({ bucketName }) => {
  const client = new S3Client({});
  const command = new PutBucketWebsiteCommand({
    Bucket: bucketName,
    WebsiteConfiguration: {
      ErrorDocument: {
        // The object key name to use when a 4XX class error occurs.
        Key: "error.html",
      },
      IndexDocument: {
        // A suffix that is appended to a request when the request is
        // for a directory.
        Suffix: "index.html",
      },
    },
  });

  try {
```

```
await client.send(command);
console.log(
  `The bucket "${bucketName}" has been configured as a static website.`
);
} catch (caught) {
  if (
    caught instanceof S3ServiceException &&
    caught.name === "NoSuchBucket"
  ) {
    console.error(
      `Error from S3 while configuring the bucket "${bucketName}" as a static
website. The bucket doesn't exist.`
    );
  } else if (caught instanceof S3ServiceException) {
    console.error(
      `Error from S3 while configuring the bucket "${bucketName}" as a static
website. ${caught.name}: ${caught.message}`
    );
  } else {
    throw caught;
  }
}
};
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [PutBucketWebsite](#) in der AWS SDK für JavaScript API-Referenz.

PutObject

Das folgende Codebeispiel zeigt die Verwendung `PutObject`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Laden Sie das Objekt hoch.

```
import { readFile } from "node:fs/promises";

import {
  PutObjectCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Upload a file to an S3 bucket.
 * @param {{ bucketName: string, key: string, filePath: string }}
 */
export const main = async ({ bucketName, key, filePath }) => {
  const client = new S3Client({});
  const command = new PutObjectCommand({
    Bucket: bucketName,
    Key: key,
    Body: await readFile(filePath),
  });

  try {
    const response = await client.send(command);
    console.log(response);
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "EntityTooLarge"
    ) {
      console.error(
        `Error from S3 while uploading object to ${bucketName}. \
The object was too large. To upload objects larger than 5GB, use the S3 console \
(160GB max) \
or the multipart upload API (5TB max).`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while uploading object to ${bucketName}. ${caught.name}: \
${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
}
```

```
};
```

Laden Sie das Objekt hoch, sofern es ETag mit dem angegebenen übereinstimmt.

```
import {
  GetObjectCommand,
  NoSuchKey,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Get a single object from a specified S3 bucket.
 * @param {{ bucketName: string, key: string, eTag: string }}
 */
export const main = async ({ bucketName, key, eTag }) => {
  const client = new S3Client({});

  try {
    const response = await client.send(
      new GetObjectCommand({
        Bucket: bucketName,
        Key: key,
        IfMatch: eTag,
      }),
    );
    // The Body object also has 'transformToByteArray' and 'transformToWebStream'
    methods.
    const str = await response.Body.transformToString();
    console.log("Success. Here is text of the file:", str);
  } catch (caught) {
    if (caught instanceof NoSuchKey) {
      console.error(
        `Error from S3 while getting object "${key}" from "${bucketName}". No such
        key exists.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while getting object from ${bucketName}. ${caught.name}:
        ${caught.message}`,
      );
    }
  }
}
```

```
    } else {
      throw caught;
    }
  }
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
  isMain,
  validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
  const options = {
    bucketName: {
      type: "string",
      required: true,
    },
    key: {
      type: "string",
      required: true,
    },
    eTag: {
      type: "string",
      required: true,
    },
  };
};

const results = parseArgs({ options });
const { errors } = validateArgs({ options }, results);
return { errors, results };
};

if (isMain(import.meta.url)) {
  const { errors, results } = loadArgs();
  if (!errors) {
    main(results.values);
  } else {
    console.error(errors.join("\n"));
  }
}
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).

- Einzelheiten zur API finden Sie [PutObject](#) in der AWS SDK für JavaScript API-Referenz.

PutObjectLegalHold

Das folgende Codebeispiel zeigt die Verwendung `PutObjectLegalHold`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import {
  PutObjectLegalHoldCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Apply a legal hold configuration to the specified object.
 * @param {{ bucketName: string, objectKey: string, legalHoldStatus: "ON" | "OFF" }}
 */
export const main = async ({ bucketName, objectKey, legalHoldStatus }) => {
  if (!["OFF", "ON"].includes(legalHoldStatus.toUpperCase())) {
    throw new Error(
      "Invalid parameter. legalHoldStatus must be 'ON' or 'OFF'."
    );
  }

  const client = new S3Client({});
  const command = new PutObjectLegalHoldCommand({
    Bucket: bucketName,
    Key: objectKey,
    LegalHold: {
      // Set the status to 'ON' to place a legal hold on the object.
      // Set the status to 'OFF' to remove the legal hold.
      Status: legalHoldStatus,
    },
  });
};
```

```
try {
  await client.send(command);
  console.log(
    `Legal hold status set to "${legalHoldStatus}" for "${objectKey}" in
"${bucketName}"`,
  );
} catch (caught) {
  if (
    caught instanceof S3ServiceException &&
    caught.name === "NoSuchBucket"
  ) {
    console.error(
      `Error from S3 while modifying legal hold status for "${objectKey}" in
"${bucketName}". The bucket doesn't exist.`,
    );
  } else if (caught instanceof S3ServiceException) {
    console.error(
      `Error from S3 while modifying legal hold status for "${objectKey}" in
"${bucketName}". ${caught.name}: ${caught.message}`,
    );
  } else {
    throw caught;
  }
}
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
  isMain,
  validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
  const options = {
    bucketName: {
      type: "string",
      required: true,
    },
    objectKey: {
      type: "string",
      required: true,
    },
    legalHoldStatus: {
```

```
        type: "string",
        default: "ON",
    },
};
const results = parseArgs({ options });
const { errors } = validateArgs({ options }, results);
return { errors, results };
};

if (isMain(import.meta.url)) {
    const { errors, results } = loadArgs();
    if (!errors) {
        main(results.values);
    } else {
        console.error(errors.join("\n"));
    }
}
```

- Einzelheiten zur API finden Sie [PutObjectLegalHold](#) in der AWS SDK für JavaScript API-Referenz.

PutObjectLockConfiguration

Das folgende Codebeispiel zeigt die Verwendung PutObjectLockConfiguration.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Legen Sie die Object-Lock-Konfiguration eines Buckets fest.

```
import {
    PutObjectLockConfigurationCommand,
    S3Client,
    S3ServiceException,
} from "@aws-sdk/client-s3";
```

```

/**
 * Enable S3 Object Lock for an Amazon S3 bucket.
 * After you enable Object Lock on a bucket, you can't
 * disable Object Lock or suspend versioning for that bucket.
 * @param {{ bucketName: string, enabled: boolean }}
 */
export const main = async ({ bucketName }) => {
  const client = new S3Client({});
  const command = new PutObjectLockConfigurationCommand({
    Bucket: bucketName,
    // The Object Lock configuration that you want to apply to the specified bucket.
    ObjectLockConfiguration: {
      ObjectLockEnabled: "Enabled",
    },
  });

  try {
    await client.send(command);
    console.log(`Object Lock for "${bucketName}" enabled.`);
  } catch (caught) {
    if (
      caught instanceof S3ServiceException &&
      caught.name === "NoSuchBucket"
    ) {
      console.error(
        `Error from S3 while modifying the object lock configuration for the bucket "${bucketName}". The bucket doesn't exist.`,
      );
    } else if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while modifying the object lock configuration for the bucket "${bucketName}". ${caught.name}: ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
  isMain,
  validateArgs,

```

```

} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
  const options = {
    bucketName: {
      type: "string",
      required: true,
    },
  };
};

const results = parseArgs({ options });
const { errors } = validateArgs({ options }, results);
return { errors, results };
};

if (isMain(import.meta.url)) {
  const { errors, results } = loadArgs();
  if (!errors) {
    main(results.values);
  } else {
    console.error(errors.join("\n"));
  }
}

```

Legen Sie die Standardaufbewahrungsfrist für einen Bucket fest.

```

import {
  PutObjectLockConfigurationCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Change the default retention settings for an object in an Amazon S3 bucket.
 * @param {{ bucketName: string, retentionDays: string }}
 */
export const main = async ({ bucketName, retentionDays }) => {
  const client = new S3Client({});

  try {
    await client.send(
      new PutObjectLockConfigurationCommand({

```

```

        Bucket: bucketName,
        // The Object Lock configuration that you want to apply to the specified
bucket.
        ObjectLockConfiguration: {
            ObjectLockEnabled: "Enabled",
            Rule: {
                // The default Object Lock retention mode and period that you want to
apply
                // to new objects placed in the specified bucket. Bucket settings
require
                // both a mode and a period. The period can be either Days or Years but
                // you must select one.
                DefaultRetention: {
                    // In governance mode, users can't overwrite or delete an object
version
                    // or alter its lock settings unless they have special permissions.
With
                    // governance mode, you protect objects against being deleted by most
users,
                    // but you can still grant some users permission to alter the
retention settings
                    // or delete the objects if necessary.
                    Mode: "GOVERNANCE",
                    Days: Number.parseInt(retentionDays),
                },
            },
        },
    });
    console.log(
        `Set default retention mode to "GOVERNANCE" with a retention period of
${retentionDays} day(s).`,
    );
} catch (caught) {
    if (
        caught instanceof S3ServiceException &&
        caught.name === "NoSuchBucket"
    ) {
        console.error(
            `Error from S3 while setting the default object retention for a bucket. The
bucket doesn't exist.`,
        );
    } else if (caught instanceof S3ServiceException) {
        console.error(

```

```
        `Error from S3 while setting the default object retention for a bucket.
        ${caught.name}: ${caught.message}` ,
        );
    } else {
        throw caught;
    }
}
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
    isMain,
    validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
    const options = {
        bucketName: {
            type: "string",
            required: true,
        },
        retentionDays: {
            type: "string",
            required: true,
        },
    };
    const results = parseArgs({ options });
    const { errors } = validateArgs({ options }, results);
    return { errors, results };
};

if (isMain(import.meta.url)) {
    const { errors, results } = loadArgs();
    if (!errors) {
        main(results.values);
    } else {
        console.error(errors.join("\n"));
    }
}
```

- Einzelheiten zur API finden Sie [PutObjectLockConfiguration](#) in der AWS SDK für JavaScript API-Referenz.

PutObjectRetention

Das folgende Codebeispiel zeigt die Verwendung `PutObjectRetention`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import {
  PutObjectRetentionCommand,
  S3Client,
  S3ServiceException,
} from "@aws-sdk/client-s3";

/**
 * Place a 24-hour retention period on an object in an Amazon S3 bucket.
 * @param {{ bucketName: string, key: string }}
 */
export const main = async ({ bucketName, key }) => {
  const client = new S3Client({});
  const command = new PutObjectRetentionCommand({
    Bucket: bucketName,
    Key: key,
    BypassGovernanceRetention: false,
    Retention: {
      // In governance mode, users can't overwrite or delete an object version
      // or alter its lock settings unless they have special permissions. With
      // governance mode, you protect objects against being deleted by most users,
      // but you can still grant some users permission to alter the retention
      settings
      // or delete the objects if necessary.
      Mode: "GOVERNANCE",
      RetainUntilDate: new Date(new Date().getTime() + 24 * 60 * 60 * 1000),
    },
  });
```



```
try {
  await client.send(command);
  console.log("Object Retention settings updated.");
} catch (caught) {
  if (
    caught instanceof S3ServiceException &&
    caught.name === "NoSuchBucket"
  ) {
    console.error(
      `Error from S3 while modifying the governance mode and retention period on
an object. The bucket doesn't exist.`,
    );
  } else if (caught instanceof S3ServiceException) {
    console.error(
      `Error from S3 while modifying the governance mode and retention period on
an object. ${caught.name}: ${caught.message}`,
    );
  } else {
    throw caught;
  }
}
};

// Call function if run directly
import { parseArgs } from "node:util";
import {
  isMain,
  validateArgs,
} from "@aws-doc-sdk-examples/lib/utils/util-node.js";

const loadArgs = () => {
  const options = {
    bucketName: {
      type: "string",
      required: true,
    },
    key: {
      type: "string",
      required: true,
    },
  };
  const results = parseArgs({ options });
  const { errors } = validateArgs({ options }, results);
```

```
    return { errors, results };
};

if (isMain(import.meta.url)) {
    const { errors, results } = loadArgs();
    if (!errors) {
        main(results.values);
    } else {
        console.error(errors.join("\n"));
    }
}
```

- Einzelheiten zur API finden Sie [PutObjectRetention](#) in der AWS SDK für JavaScript API-Referenz.

Szenarien

Eine vorsignierte URL erstellen

Das folgende Codebeispiel veranschaulicht, wie Sie eine vorsignierte URL für Amazon S3 erstellen und ein Objekt hochladen.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie eine vorsignierte URL, um ein Objekt in einen Bucket hochzuladen.

```
import https from "node:https";

import { XMLParser } from "fast-xml-parser";
import { PutObjectCommand, S3Client } from "@aws-sdk/client-s3";
import { fromIni } from "@aws-sdk/credential-providers";
import { HttpRequest } from "@smithy/protocol-http";
import {
    getSignedUrl,
```

```

    S3RequestPresigner,
  } from "@aws-sdk/s3-request-presigner";
import { parseUrl } from "@smithy/url-parser";
import { formatUrl } from "@aws-sdk/util-format-url";
import { Hash } from "@smithy/hash-node";

const createPresignedUrlWithoutClient = async ({ region, bucket, key }) => {
  const url = parseUrl(`https://${bucket}.s3.${region}.amazonaws.com/${key}`);
  const presigner = new S3RequestPresigner({
    credentials: fromIni(),
    region,
    sha256: Hash.bind(null, "sha256"),
  });

  const signedUrlObject = await presigner.presign(
    new HttpRequest({ ...url, method: "PUT" }),
  );
  return formatUrl(signedUrlObject);
};

const createPresignedUrlWithClient = ({ region, bucket, key }) => {
  const client = new S3Client({ region });
  const command = new PutObjectCommand({ Bucket: bucket, Key: key });
  return getSignedUrl(client, command, { expiresIn: 3600 });
};

/**
 * Make a PUT request to the provided URL.
 *
 * @param {string} url
 * @param {string} data
 */
const put = (url, data) => {
  return new Promise((resolve, reject) => {
    const req = https.request(
      url,
      { method: "PUT", headers: { "Content-Length": new Blob([data]).size } },
      (res) => {
        let responseBody = "";
        res.on("data", (chunk) => {
          responseBody += chunk;
        });
        res.on("end", () => {
          const parser = new XMLParser();

```

```
        if (res.statusCode >= 200 && res.statusCode <= 299) {
            resolve(parser.parse(responseBody, true));
        } else {
            reject(parser.parse(responseBody, true));
        }
    });
},
);
req.on("error", (err) => {
    reject(err);
});
req.write(data);
req.end();
});
};

/**
 * Create two presigned urls for uploading an object to an S3 bucket.
 * The first presigned URL is created with credentials from the shared INI file
 * in the current environment. The second presigned URL is created using an
 * existing S3Client instance that has already been provided with credentials.
 * @param {{ bucketName: string, key: string, region: string }}
 */
export const main = async ({ bucketName, key, region }) => {
    try {
        const noClientUrl = await createPresignedUrlWithoutClient({
            bucket: bucketName,
            key,
            region,
        });

        const clientUrl = await createPresignedUrlWithClient({
            bucket: bucketName,
            region,
            key,
        });

        // After you get the presigned URL, you can provide your own file
        // data. Refer to put() above.
        console.log("Calling PUT using presigned URL without client");
        await put(noClientUrl, "Hello World");

        console.log("Calling PUT using presigned URL with client");
        await put(clientUrl, "Hello World");
    }
}
```

```

    console.log("\nDone. Check your S3 console.");
  } catch (caught) {
    if (caught instanceof Error && caught.name === "CredentialsProviderError") {
      console.error(
        `There was an error getting your credentials. Are your local credentials
configured?\n${caught.name}: ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};

```

Erstellen Sie eine vorsignierte URL, um ein Objekt aus einem Bucket herunterzuladen.

```

import { GetObjectCommand, S3Client } from "@aws-sdk/client-s3";
import { fromIni } from "@aws-sdk/credential-providers";
import { HttpRequest } from "@smithy/protocol-http";
import {
  getSignedUrl,
  S3RequestPresigner,
} from "@aws-sdk/s3-request-presigner";
import { parseUrl } from "@smithy/url-parser";
import { formatUrl } from "@aws-sdk/util-format-url";
import { Hash } from "@smithy/hash-node";

const createPresignedUrlWithoutClient = async ({ region, bucket, key }) => {
  const url = parseUrl(`https://${bucket}.s3.${region}.amazonaws.com/${key}`);
  const presigner = new S3RequestPresigner({
    credentials: fromIni(),
    region,
    sha256: Hash.bind(null, "sha256"),
  });

  const signedUrlObject = await presigner.presign(new HttpRequest(url));
  return formatUrl(signedUrlObject);
};

const createPresignedUrlWithClient = ({ region, bucket, key }) => {
  const client = new S3Client({ region });
  const command = new GetObjectCommand({ Bucket: bucket, Key: key });

```

```

    return getSignedUrl(client, command, { expiresIn: 3600 });
};

/**
 * Create two presigned urls for downloading an object from an S3 bucket.
 * The first presigned URL is created with credentials from the shared INI file
 * in the current environment. The second presigned URL is created using an
 * existing S3Client instance that has already been provided with credentials.
 * @param {{ bucketName: string, key: string, region: string }}
 */
export const main = async ({ bucketName, key, region }) => {
  try {
    const noClientUrl = await createPresignedUrlWithoutClient({
      bucket: bucketName,
      region,
      key,
    });

    const clientUrl = await createPresignedUrlWithClient({
      bucket: bucketName,
      region,
      key,
    });

    console.log("Presigned URL without client");
    console.log(noClientUrl);
    console.log("\n");

    console.log("Presigned URL with client");
    console.log(clientUrl);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "CredentialsProviderError") {
      console.error(
        `There was an error getting your credentials. Are your local credentials configured?\n${caught.name}: ${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
};

```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).

Erstellen einer Serverless-Anwendung zur Verwaltung von Fotos

Das folgende Codebeispiel zeigt, wie eine Serverless-Anwendung erstellt wird, mit der Benutzer Fotos mithilfe von Labels erstellen können.

SDK für JavaScript (v3)

Zeigt, wie eine Anwendung zur Verwaltung von Fotobeständen entwickelt wird, die mithilfe von Amazon Rekognition Labels in Bildern erkennt und sie für einen späteren Abruf speichert.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

Einen tiefen Einblick in den Ursprung dieses Beispiels finden Sie im Beitrag in der [AWS - Community](#).

In diesem Beispiel verwendete Dienste

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Eine Webseite erstellen, die Amazon-S3-Objekte auflistet

Im folgenden Codebeispiel wird veranschaulicht, wie Sie Amazon-S3-Objekte auf einer Webseite auflisten.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Der folgende Code ist die relevante React-Komponente, die das AWS SDK aufruft. Eine lauffähige Version der Anwendung, die diese Komponente enthält, finden Sie unter dem vorherigen GitHub Link.

```
import { useEffect, useState } from "react";
import {
  ListObjectsCommand,
  type ListObjectsCommandOutput,
  S3Client,
} from "@aws-sdk/client-s3";
import { fromCognitoIdentityPool } from "@aws-sdk/credential-providers";
import "./App.css";

function App() {
  const [objects, setObjects] = useState<
    Required<ListObjectsCommandOutput>["Contents"]
  >([]);

  useEffect(() => {
    const client = new S3Client({
      region: "us-east-1",
      // Unless you have a public bucket, you'll need access to a private bucket.
      // One way to do this is to create an Amazon Cognito identity pool, attach a
      // role to the pool,
      // and grant the role access to the 's3:GetObject' action.
      //
      // You'll also need to configure the CORS settings on the bucket to allow
      // traffic from
      // this example site. Here's an example configuration that allows all origins.
      // Don't
      // do this in production.
      // [
      //   {
      //     "AllowedHeaders": ["*"],
      //     "AllowedMethods": ["GET"],
      //     "AllowedOrigins": ["*"],
      //     "ExposeHeaders": [],
      //   },
      // ],
      //
      credentials: fromCognitoIdentityPool({
        clientConfig: { region: "us-east-1" },
        identityPoolId: "<YOUR_IDENTITY_POOL_ID>",
      })
    });
```



```
    }},
  });
  const command = new ListObjectsCommand({ Bucket: "bucket-name" });
  client.send(command).then(({ Contents }) => setObjects(Contents || []));
}, []);

return (
  <div className="App">
    {objects.map((o) => (
      <div key={o.ETag}>{o.Key}</div>
    ))}
  </div>
);
}

export default App;
```

- Einzelheiten zur API finden Sie unter [ListObjects AWS SDK für JavaScript](#) API-Referenz.

Erstellen Sie eine Amazon-Texttract-Explorer-Anwendung

Die folgenden Codebeispiele zeigen, wie die Amazon-Texttract-Ausgabe in einer interaktiven Anwendung untersucht werden kann.

SDK für JavaScript (v3)

Zeigt, wie Sie mit AWS SDK für JavaScript dem eine React-Anwendung erstellen, die Amazon Texttract verwendet, um Daten aus einem Dokumentbild zu extrahieren und auf einer interaktiven Webseite anzuzeigen. Dieses Beispiel wird in einem Webbrowser ausgeführt und erfordert eine authentifizierte Amazon-Cognito-Identität für Anmeldeinformationen. Es verwendet Amazon Simple Storage Service (Amazon S3) zur Speicherung und fragt für Benachrichtigungen eine Amazon Simple Queue Service (Amazon SQS)-Warteschlange ab, die ein Amazon Simple Notification Service (Amazon SNS)-Thema abonniert hat.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

In diesem Beispiel verwendete Dienste

- Amazon Cognito Identity
- Amazon S3

- Amazon SNS
- Amazon SQS
- Amazon Textract

Löschen aller Objekte in einem Bucket

Im folgenden Codebeispiel wird gezeigt, wie Sie alle Objekte aus einem Amazon-S3-Bucket löschen.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Löschen Sie alle Objekte für einen gegebenen Amazon-S3-Bucket.

```
import {
  DeleteObjectsCommand,
  paginateListObjectsV2,
  S3Client,
} from "@aws-sdk/client-s3";

/**
 *
 * @param {{ bucketName: string }} config
 */
export const main = async ({ bucketName }) => {
  const client = new S3Client({});
  try {
    console.log(`Deleting all objects in bucket: ${bucketName}`);

    const paginator = paginateListObjectsV2(
      { client },
      {
        Bucket: bucketName,
      },
    );
  }
};
```

```
const objectKeys = [];
for await (const { Contents } of paginator) {
  objectKeys.push(...Contents.map((obj) => ({ Key: obj.Key })));
}

const deleteCommand = new DeleteObjectsCommand({
  Bucket: bucketName,
  Delete: { Objects: objectKeys },
});

await client.send(deleteCommand);

console.log(`All objects deleted from bucket: ${bucketName}`);
} catch (caught) {
  if (caught instanceof Error) {
    console.error(
      `Failed to empty ${bucketName}. ${caught.name}: ${caught.message}`,
    );
  }
}
};

// Call function if run directly.
import { fileURLToPath } from "node:url";
import { parseArgs } from "node:util";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const options = {
    bucketName: {
      type: "string",
    },
  };
};

const { values } = parseArgs({ options });
main(values);
}
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für JavaScript - API-Referenz.
 - [DeleteObjects](#)
 - [ListObjectsV2](#)

Erkennen von Objekten in Bildern

Die folgenden Codebeispiele zeigen, wie man eine App erstellt, die Amazon Rekognition verwendet, um Objekte nach Kategorien in Bildern zu erkennen.

SDK für JavaScript (v3)

Zeigt, wie Amazon Rekognition zusammen mit dem verwendet wird, um eine App AWS SDK für JavaScript zu erstellen, die Amazon Rekognition verwendet, um Objekte nach Kategorien in Bildern zu identifizieren, die sich in einem Amazon Simple Storage Service (Amazon S3) -Bucket befinden. Die App sendet dem Administrator eine E-Mail-Benachrichtigung mit den Ergebnissen über Amazon Simple Email Service (Amazon SES).

Lernen Sie Folgendes:

- Erstellen Sie mit Amazon Cognito einen nicht authentifizierten Benutzer.
- Analysieren Sie mit Amazon Rekognition Bilder für Objekte.
- Verifizieren Sie eine E-Mail-Adresse für Amazon SES.
- Senden Sie eine E-Mail-Benachrichtigung mit Amazon SES.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter. [GitHub](#)

In diesem Beispiel verwendete Dienste

- Amazon Rekognition
- Amazon S3
- Amazon SES

Sperren von Amazon-S3-Objekten

Das folgende Codebeispiel zeigt, wie Sie mit S3-Object-Lock-Features arbeiten.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Einstiegspunkt für das Szenario (index.js). Dadurch werden alle Schritte orchestriert. Die Implementierungsdetails für Szenario, ScenarioInput ScenarioOutput, und finden GitHub Sie unter ScenarioAction.

```
import * as Scenarios from "@aws-doc-sdk-examples/lib/scenario/index.js";
import {
  exitOnFalse,
  loadState,
  saveState,
} from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";

import { welcome, welcomeContinue } from "../welcome.steps.js";
import {
  confirmCreateBuckets,
  confirmPopulateBuckets,
  confirmSetLegalHoldFileEnabled,
  confirmSetLegalHoldFileRetention,
  confirmSetRetentionPeriodFileEnabled,
  confirmSetRetentionPeriodFileRetention,
  confirmUpdateLockPolicy,
  confirmUpdateRetention,
  createBuckets,
  createBucketsAction,
  getBucketPrefix,
  populateBuckets,
  populateBucketsAction,
  setLegalHoldFileEnabledAction,
  setLegalHoldFileRetentionAction,
  setRetentionPeriodFileEnabledAction,
  setRetentionPeriodFileRetentionAction,
  updateLockPolicy,
  updateLockPolicyAction,
  updateRetention,
  updateRetentionAction,
} from "../setup.steps.js";

/**
 * @param {Scenarios} scenarios
 * @param {Record<string, any>} initialState
 */
export const getWorkflowStages = (scenarios, initialState = {}) => {
  const client = new S3Client({});
```

```
return {
  deploy: new scenarios.Scenario(
    "S3 Object Locking - Deploy",
    [
      welcome(scenarios),
      welcomeContinue(scenarios),
      exitOnFalse(scenarios, "welcomeContinue"),
      getBucketPrefix(scenarios),
      createBuckets(scenarios),
      confirmCreateBuckets(scenarios),
      exitOnFalse(scenarios, "confirmCreateBuckets"),
      createBucketsAction(scenarios, client),
      updateRetention(scenarios),
      confirmUpdateRetention(scenarios),
      exitOnFalse(scenarios, "confirmUpdateRetention"),
      updateRetentionAction(scenarios, client),
      populateBuckets(scenarios),
      confirmPopulateBuckets(scenarios),
      exitOnFalse(scenarios, "confirmPopulateBuckets"),
      populateBucketsAction(scenarios, client),
      updateLockPolicy(scenarios),
      confirmUpdateLockPolicy(scenarios),
      exitOnFalse(scenarios, "confirmUpdateLockPolicy"),
      updateLockPolicyAction(scenarios, client),
      confirmSetLegalHoldFileEnabled(scenarios),
      setLegalHoldFileEnabledAction(scenarios, client),
      confirmSetRetentionPeriodFileEnabled(scenarios),
      setRetentionPeriodFileEnabledAction(scenarios, client),
      confirmSetLegalHoldFileRetention(scenarios),
      setLegalHoldFileRetentionAction(scenarios, client),
      confirmSetRetentionPeriodFileRetention(scenarios),
      setRetentionPeriodFileRetentionAction(scenarios, client),
      saveState,
    ],
    initialState,
  ),
  demo: new scenarios.Scenario(
    "S3 Object Locking - Demo",
    [loadState, replAction(scenarios, client)],
    initialState,
  ),
  clean: new scenarios.Scenario(
    "S3 Object Locking - Destroy",
    [
```

```

        loadState,
        confirmCleanup(scenarios),
        exitOnFalse(scenarios, "confirmCleanup"),
        cleanupAction(scenarios, client),
    ],
    initialState,
),
};
};

// Call function if run directly
import { fileURLToPath } from "node:url";
import { S3Client } from "@aws-sdk/client-s3";
import { cleanupAction, confirmCleanup } from "./clean.steps.js";
import { replAction } from "./repl.steps.js";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
    const objectLockingScenarios = getWorkflowStages(Scenarios);
    Scenarios.parseScenarioArgs(objectLockingScenarios, {
        name: "Amazon S3 object locking workflow",
        description:
            "Work with Amazon Simple Storage Service (Amazon S3) object locking",
        features: "",
        synopsis:
            "node index.js --scenario <deploy | demo | clean> [-h|--help] [-y|--yes] [-v|--verbose]",
    });
}

```

Geben Sie Willkommensnachrichten an die Konsole aus (welcome.steps.js).

```

/**
 * @typedef {import("@aws-doc-sdk-examples/lib/scenario/index.js")} Scenarios
 */

/**
 * @param {Scenarios} scenarios
 */
const welcome = (scenarios) =>
    new scenarios.ScenarioOutput(
        "welcome",

```

```

    "Welcome to the Amazon Simple Storage Service (S3) Object Locking Feature
    Scenario. For this workflow, we will use the AWS SDK for JavaScript to create
    several S3 buckets and files to demonstrate working with S3 locking features.",
    { header: true },
  );

/**
 * @param {Scenarios} scenarios
 */
const welcomeContinue = (scenarios) =>
  new scenarios.ScenarioInput(
    "welcomeContinue",
    "Press Enter when you are ready to start.",
    { type: "confirm" },
  );

export { welcome, welcomeContinue };

```

Stellen Sie Buckets, Objekte und Dateieinstellungen bereit (setup.steps.js).

```

import {
  BucketVersioningStatus,
  ChecksumAlgorithm,
  CreateBucketCommand,
  MFADeleteStatus,
  PutBucketVersioningCommand,
  PutObjectCommand,
  PutObjectLockConfigurationCommand,
  PutObjectLegalHoldCommand,
  PutObjectRetentionCommand,
  ObjectLockLegalHoldStatus,
  ObjectLockRetentionMode,
  GetBucketVersioningCommand,
  BucketAlreadyExists,
  BucketAlreadyOwnedByYou,
  S3ServiceException,
  waitUntilBucketExists,
} from "@aws-sdk/client-s3";

import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

/**

```



```

* @typedef {import("@aws-doc-sdk-examples/lib/scenario/index.js")} Scenarios
*/

/**
* @typedef {import("@aws-sdk/client-s3").S3Client} S3Client
*/

/**
* @param {Scenarios} scenarios
*/
const getBucketPrefix = (scenarios) =>
  new scenarios.ScenarioInput(
    "bucketPrefix",
    "Provide a prefix that will be used for bucket creation.",
    { type: "input", default: "amzn-s3-demo-bucket" },
  );

/**
* @param {Scenarios} scenarios
*/
const createBuckets = (scenarios) =>
  new scenarios.ScenarioOutput(
    "createBuckets",
    (state) => `The following buckets will be created:
      ${state.bucketPrefix}-no-lock with object lock False.
      ${state.bucketPrefix}-lock-enabled with object lock True.
      ${state.bucketPrefix}-retention-after-creation with object lock False.`,
    { preformatted: true },
  );

/**
* @param {Scenarios} scenarios
*/
const confirmCreateBuckets = (scenarios) =>
  new scenarios.ScenarioInput("confirmCreateBuckets", "Create the buckets?", {
    type: "confirm",
  });

/**
* @param {Scenarios} scenarios
* @param {S3Client} client
*/
const createBucketsAction = (scenarios, client) =>
  new scenarios.ScenarioAction("createBucketsAction", async (state) => {

```

```

const noLockBucketName = `${state.bucketPrefix}-no-lock`;
const lockEnabledBucketName = `${state.bucketPrefix}-lock-enabled`;
const retentionBucketName = `${state.bucketPrefix}-retention-after-creation`;

try {
  await client.send(new CreateBucketCommand({ Bucket: noLockBucketName }));
  await waitUntilBucketExists({ client }, { Bucket: noLockBucketName });
  await client.send(
    new CreateBucketCommand({
      Bucket: lockEnabledBucketName,
      ObjectLockEnabledForBucket: true,
    }),
  );
  await waitUntilBucketExists(
    { client },
    { Bucket: lockEnabledBucketName },
  );
  await client.send(
    new CreateBucketCommand({ Bucket: retentionBucketName }),
  );
  await waitUntilBucketExists({ client }, { Bucket: retentionBucketName });

  state.noLockBucketName = noLockBucketName;
  state.lockEnabledBucketName = lockEnabledBucketName;
  state.retentionBucketName = retentionBucketName;
} catch (caught) {
  if (
    caught instanceof BucketAlreadyExists ||
    caught instanceof BucketAlreadyOwnedByYou
  ) {
    console.error(`${caught.name}: ${caught.message}`);
    state.earlyExit = true;
  } else {
    throw caught;
  }
}
});

/**
 * @param {Scenarios} scenarios
 */
const populateBuckets = (scenarios) =>
  new scenarios.ScenarioOutput(
    "populateBuckets",

```

```

    (state) => `The following test files will be created:
      file0.txt in ${state.bucketPrefix}-no-lock.
      file1.txt in ${state.bucketPrefix}-no-lock.
      file0.txt in ${state.bucketPrefix}-lock-enabled.
      file1.txt in ${state.bucketPrefix}-lock-enabled.
      file0.txt in ${state.bucketPrefix}-retention-after-creation.
      file1.txt in ${state.bucketPrefix}-retention-after-creation.` ,
    { preformatted: true },
  );

/**
 * @param {Scenarios} scenarios
 */
const confirmPopulateBuckets = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmPopulateBuckets",
    "Populate the buckets?",
    { type: "confirm" },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const populateBucketsAction = (scenarios, client) =>
  new scenarios.ScenarioAction("populateBucketsAction", async (state) => {
    try {
      await client.send(
        new PutObjectCommand({
          Bucket: state.noLockBucketName,
          Key: "file0.txt",
          Body: "Content",
          ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
        }),
      );
      await client.send(
        new PutObjectCommand({
          Bucket: state.noLockBucketName,
          Key: "file1.txt",
          Body: "Content",
          ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
        }),
      );
      await client.send(

```

```

        new PutObjectCommand({
            Bucket: state.lockEnabledBucketName,
            Key: "file0.txt",
            Body: "Content",
            ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
        }),
    );
    await client.send(
        new PutObjectCommand({
            Bucket: state.lockEnabledBucketName,
            Key: "file1.txt",
            Body: "Content",
            ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
        }),
    );
    await client.send(
        new PutObjectCommand({
            Bucket: state.retentionBucketName,
            Key: "file0.txt",
            Body: "Content",
            ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
        }),
    );
    await client.send(
        new PutObjectCommand({
            Bucket: state.retentionBucketName,
            Key: "file1.txt",
            Body: "Content",
            ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
        }),
    );
} catch (caught) {
    if (caught instanceof S3ServiceException) {
        console.error(
            `Error from S3 while uploading object.  ${caught.name}:
${caught.message}`,
        );
    } else {
        throw caught;
    }
}
});

```

/**

```

* @param {Scenarios} scenarios
*/
const updateRetention = (scenarios) =>
  new scenarios.ScenarioOutput(
    "updateRetention",
    (state) => `A bucket can be configured to use object locking with a default
  retention period.
  A default retention period will be configured for ${state.bucketPrefix}-retention-
  after-creation.` ,
    { preformatted: true },
  );

/**
* @param {Scenarios} scenarios
*/
const confirmUpdateRetention = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmUpdateRetention",
    "Configure default retention period?",
    { type: "confirm" },
  );

/**
* @param {Scenarios} scenarios
* @param {S3Client} client
*/
const updateRetentionAction = (scenarios, client) =>
  new scenarios.ScenarioAction("updateRetentionAction", async (state) => {
    await client.send(
      new PutBucketVersioningCommand({
        Bucket: state.retentionBucketName,
        VersioningConfiguration: {
          MFADelete: MFADeleteStatus.Disabled,
          Status: BucketVersioningStatus.Enabled,
        },
      }),
    );

    const getBucketVersioning = new GetBucketVersioningCommand({
      Bucket: state.retentionBucketName,
    });

    await retry({ intervalInMs: 500, maxRetries: 10 }, async () => {
      const { Status } = await client.send(getBucketVersioning);
    });
  });

```

```

    if (Status !== "Enabled") {
      throw new Error("Bucket versioning is not enabled.");
    }
  });

  await client.send(
    new PutObjectLockConfigurationCommand({
      Bucket: state.retentionBucketName,
      ObjectLockConfiguration: {
        ObjectLockEnabled: "Enabled",
        Rule: {
          DefaultRetention: {
            Mode: "GOVERNANCE",
            Years: 1,
          },
        },
      },
    }),
  );
});

/**
 * @param {Scenarios} scenarios
 */
const updateLockPolicy = (scenarios) =>
  new scenarios.ScenarioOutput(
    "updateLockPolicy",
    (state) => `Object lock policies can also be added to existing buckets.
An object lock policy will be added to ${state.bucketPrefix}-lock-enabled.`,
    { preformatted: true },
  );

/**
 * @param {Scenarios} scenarios
 */
const confirmUpdateLockPolicy = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmUpdateLockPolicy",
    "Add object lock policy?",
    { type: "confirm" },
  );

/**
 * @param {Scenarios} scenarios

```

```

* @param {S3Client} client
*/
const updateLockPolicyAction = (scenarios, client) =>
  new scenarios.ScenarioAction("updateLockPolicyAction", async (state) => {
    await client.send(
      new PutObjectLockConfigurationCommand({
        Bucket: state.lockEnabledBucketName,
        ObjectLockConfiguration: {
          ObjectLockEnabled: "Enabled",
        },
      }),
    );
  });

/**
* @param {Scenarios} scenarios
* @param {S3Client} client
*/
const confirmSetLegalHoldFileEnabled = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmSetLegalHoldFileEnabled",
    (state) =>
      `Would you like to add a legal hold to file0.txt in
${state.lockEnabledBucketName}?`,
    {
      type: "confirm",
    },
  );

/**
* @param {Scenarios} scenarios
* @param {S3Client} client
*/
const setLegalHoldFileEnabledAction = (scenarios, client) =>
  new scenarios.ScenarioAction(
    "setLegalHoldFileEnabledAction",
    async (state) => {
      await client.send(
        new PutObjectLegalHoldCommand({
          Bucket: state.lockEnabledBucketName,
          Key: "file0.txt",
          LegalHold: {
            Status: ObjectLockLegalHoldStatus.ON,
          },
        }),
      );
    },
  );

```

```

    })),
  );
  console.log(
    `Modified legal hold for file0.txt in ${state.lockEnabledBucketName}.`,
  );
},
{ skipWhen: (state) => !state.confirmSetLegalHoldFileEnabled },
);

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const confirmSetRetentionPeriodFileEnabled = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmSetRetentionPeriodFileEnabled",
    (state) =>
      `Would you like to add a 1 day Governance retention period to file1.txt in
      ${state.lockEnabledBucketName}?
      Reminder: Only a user with the s3:BypassGovernanceRetention permission will be able
      to delete this file or its bucket until the retention period has expired.`,
    {
      type: "confirm",
    },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const setRetentionPeriodFileEnabledAction = (scenarios, client) =>
  new scenarios.ScenarioAction(
    "setRetentionPeriodFileEnabledAction",
    async (state) => {
      const retentionDate = new Date();
      retentionDate.setDate(retentionDate.getDate() + 1);
      await client.send(
        new PutObjectRetentionCommand({
          Bucket: state.lockEnabledBucketName,
          Key: "file1.txt",
          Retention: {
            Mode: ObjectLockRetentionMode.GOVERNANCE,
            RetainUntilDate: retentionDate,
          },
        },

```



```

    })),
  );
  console.log(
    `Set retention for file1.txt in ${state.lockEnabledBucketName} until
    ${retentionDate.toISOString().split("T")[0]}.`,
  );
},
{ skipWhen: (state) => !state.confirmSetRetentionPeriodFileEnabled },
);

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const confirmSetLegalHoldFileRetention = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmSetLegalHoldFileRetention",
    (state) =>
      `Would you like to add a legal hold to file0.txt in
      ${state.retentionBucketName}?`,
    {
      type: "confirm",
    },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const setLegalHoldFileRetentionAction = (scenarios, client) =>
  new scenarios.ScenarioAction(
    "setLegalHoldFileRetentionAction",
    async (state) => {
      await client.send(
        new PutObjectLegalHoldCommand({
          Bucket: state.retentionBucketName,
          Key: "file0.txt",
          LegalHold: {
            Status: ObjectLockLegalHoldStatus.ON,
          },
        }),
      );
      console.log(
        `Modified legal hold for file0.txt in ${state.retentionBucketName}.`,
      );
    },
  );

```

```

    );
  },
  { skipWhen: (state) => !state.confirmSetLegalHoldFileRetention },
);

/**
 * @param {Scenarios} scenarios
 */
const confirmSetRetentionPeriodFileRetention = (scenarios) =>
  new scenarios.ScenarioInput(
    "confirmSetRetentionPeriodFileRetention",
    (state) =>
      `Would you like to add a 1 day Governance retention period to file1.txt in
      ${state.retentionBucketName}?
      Reminder: Only a user with the s3:BypassGovernanceRetention permission will be able
      to delete this file or its bucket until the retention period has expired.`,
    {
      type: "confirm",
    },
  );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const setRetentionPeriodFileRetentionAction = (scenarios, client) =>
  new scenarios.ScenarioAction(
    "setRetentionPeriodFileRetentionAction",
    async (state) => {
      const retentionDate = new Date();
      retentionDate.setDate(retentionDate.getDate() + 1);
      await client.send(
        new PutObjectRetentionCommand({
          Bucket: state.retentionBucketName,
          Key: "file1.txt",
          Retention: {
            Mode: ObjectLockRetentionMode.GOVERNANCE,
            RetainUntilDate: retentionDate,
          },
          BypassGovernanceRetention: true,
        }),
      );
      console.log(

```

```
        `Set retention for file1.txt in ${state.retentionBucketName} until
        ${retentionDate.toISOString().split("T")[0]}.`,
        );
    },
    { skipWhen: (state) => !state.confirmSetRetentionPeriodFileRetention },
  );

export {
  getBucketPrefix,
  createBuckets,
  confirmCreateBuckets,
  createBucketsAction,
  populateBuckets,
  confirmPopulateBuckets,
  populateBucketsAction,
  updateRetention,
  confirmUpdateRetention,
  updateRetentionAction,
  updateLockPolicy,
  confirmUpdateLockPolicy,
  updateLockPolicyAction,
  confirmSetLegalHoldFileEnabled,
  setLegalHoldFileEnabledAction,
  confirmSetRetentionPeriodFileEnabled,
  setRetentionPeriodFileEnabledAction,
  confirmSetLegalHoldFileRetention,
  setLegalHoldFileRetentionAction,
  confirmSetRetentionPeriodFileRetention,
  setRetentionPeriodFileRetentionAction,
};
```

Zeigen Sie Dateien in den Buckets an und löschen Sie sie (repl.steps.js).

```
import {
  ChecksumAlgorithm,
  DeleteObjectCommand,
  GetObjectLegalHoldCommand,
  GetObjectLockConfigurationCommand,
  GetObjectRetentionCommand,
  ListObjectVersionsCommand,
  PutObjectCommand,
} from "@aws-sdk/client-s3";
```

```
/**
 * @typedef {import("@aws-doc-sdk-examples/lib/scenario/index.js")} Scenarios
 */

/**
 * @typedef {import("@aws-sdk/client-s3").S3Client} S3Client
 */

const choices = {
  EXIT: 0,
  LIST_ALL_FILES: 1,
  DELETE_FILE: 2,
  DELETE_FILE_WITH_RETENTION: 3,
  OVERWRITE_FILE: 4,
  VIEW_RETENTION_SETTINGS: 5,
  VIEW_LEGAL_HOLD_SETTINGS: 6,
};

/**
 * @param {Scenarios} scenarios
 */
const replInput = (scenarios) =>
  new scenarios.ScenarioInput(
    "replChoice",
    "Explore the S3 locking features by selecting one of the following choices",
    {
      type: "select",
      choices: [
        { name: "List all files in buckets", value: choices.LIST_ALL_FILES },
        { name: "Attempt to delete a file.", value: choices.DELETE_FILE },
        {
          name: "Attempt to delete a file with retention period bypass.",
          value: choices.DELETE_FILE_WITH_RETENTION,
        },
        { name: "Attempt to overwrite a file.", value: choices.OVERWRITE_FILE },
        {
          name: "View the object and bucket retention settings for a file.",
          value: choices.VIEW_RETENTION_SETTINGS,
        },
        {
          name: "View the legal hold settings for a file.",
          value: choices.VIEW_LEGAL_HOLD_SETTINGS,
        },
      ],
    },
  ),
```

```

        { name: "Finish the workflow.", value: choices.EXIT },
      ],
    },
  );

/**
 * @param {S3Client} client
 * @param {string[]} buckets
 */
const getAllFiles = async (client, buckets) => {
  /** @type {{bucket: string, key: string, version: string}[]} */
  const files = [];
  for (const bucket of buckets) {
    const objectsResponse = await client.send(
      new ListObjectVersionsCommand({ Bucket: bucket }),
    );
    for (const version of objectsResponse.Versions || []) {
      const { Key, VersionId } = version;
      files.push({ bucket, key: Key, version: VersionId });
    }
  }

  return files;
};

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const replAction = (scenarios, client) =>
  new scenarios.ScenarioAction(
    "replAction",
    async (state) => {
      const files = await getAllFiles(client, [
        state.noLockBucketName,
        state.lockEnabledBucketName,
        state.retentionBucketName,
      ]);

      const fileInput = new scenarios.ScenarioInput(
        "selectedFile",
        "Select a file:",
        {
          type: "select",

```

```

        choices: files.map((file, index) => ({
            name: `${index + 1}: ${file.bucket}: ${file.key} (version: ${
                file.version
            })`,
            value: index,
        })),
    },
);

const { replChoice } = state;

switch (replChoice) {
    case choices.LIST_ALL_FILES: {
        const files = await getAllFiles(client, [
            state.noLockBucketName,
            state.lockEnabledBucketName,
            state.retentionBucketName,
        ]);
        state.replOutput = files
            .map(
                (file) =>
                    `${file.bucket}: ${file.key} (version: ${file.version})`,
            )
            .join("\n");
        break;
    }
    case choices.DELETE_FILE: {
        /** @type {number} */
        const fileToDelete = await fileInput.handle(state);
        const selectedFile = files[fileToDelete];
        try {
            await client.send(
                new DeleteObjectCommand({
                    Bucket: selectedFile.bucket,
                    Key: selectedFile.key,
                    VersionId: selectedFile.version,
                }),
            );
            state.replOutput = `Deleted ${selectedFile.key} in
${selectedFile.bucket}.`;
        } catch (err) {
            state.replOutput = `Unable to delete object ${selectedFile.key} in
bucket ${selectedFile.bucket}: ${err.message}`;
        }
    }
}

```

```
        break;
    }
    case choices.DELETE_FILE_WITH_RETENTION: {
        /** @type {number} */
        const fileToDelete = await fileInput.handle(state);
        const selectedFile = files[fileToDelete];
        try {
            await client.send(
                new DeleteObjectCommand({
                    Bucket: selectedFile.bucket,
                    Key: selectedFile.key,
                    VersionId: selectedFile.version,
                    BypassGovernanceRetention: true,
                }),
            );
            state replOutput = `Deleted ${selectedFile.key} in
${selectedFile.bucket}.`;
        } catch (err) {
            state replOutput = `Unable to delete object ${selectedFile.key} in
bucket ${selectedFile.bucket}: ${err.message}`;
        }
        break;
    }
    case choices.OVERWRITE_FILE: {
        /** @type {number} */
        const fileToOverwrite = await fileInput.handle(state);
        const selectedFile = files[fileToOverwrite];
        try {
            await client.send(
                new PutObjectCommand({
                    Bucket: selectedFile.bucket,
                    Key: selectedFile.key,
                    Body: "New content",
                    ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
                }),
            );
            state replOutput = `Overwrote ${selectedFile.key} in
${selectedFile.bucket}.`;
        } catch (err) {
            state replOutput = `Unable to overwrite object ${selectedFile.key} in
bucket ${selectedFile.bucket}: ${err.message}`;
        }
        break;
    }
}
```

```

    case choices.VIEW_RETENTION_SETTINGS: {
      /** @type {number} */
      const fileToView = await fileInput.handle(state);
      const selectedFile = files[fileToView];
      try {
        const retention = await client.send(
          new GetObjectRetentionCommand({
            Bucket: selectedFile.bucket,
            Key: selectedFile.key,
            VersionId: selectedFile.version,
          }),
        );
        const bucketConfig = await client.send(
          new GetObjectLockConfigurationCommand({
            Bucket: selectedFile.bucket,
          }),
        );
        state.replOutput = `Object retention for ${selectedFile.key}
in ${selectedFile.bucket}: ${retention.Retention?.Mode} until
${retention.Retention?.RetainUntilDate?.toISOString()}.
Bucket object lock config for ${selectedFile.bucket} in ${selectedFile.bucket}:
Enabled: ${bucketConfig.ObjectLockConfiguration?.ObjectLockEnabled}
Rule:
${JSON.stringify(bucketConfig.ObjectLockConfiguration?.Rule?.DefaultRetention)}`;
      } catch (err) {
        state.replOutput = `Unable to fetch object lock retention:
'${err.message}'`;
      }
      break;
    }
    case choices.VIEW_LEGAL_HOLD_SETTINGS: {
      /** @type {number} */
      const fileToView = await fileInput.handle(state);
      const selectedFile = files[fileToView];
      try {
        const legalHold = await client.send(
          new GetObjectLegalHoldCommand({
            Bucket: selectedFile.bucket,
            Key: selectedFile.key,
            VersionId: selectedFile.version,
          }),
        );
        state.replOutput = `Object legal hold for ${selectedFile.key} in
${selectedFile.bucket}: Status: ${legalHold.LegalHold?.Status}`;
      }
    }
  }

```



```

        } catch (err) {
            state.replOutput = `Unable to fetch legal hold: '${err.message}'`;
        }
        break;
    }
    default:
        throw new Error(`Invalid replChoice: ${replChoice}`);
    }
},
{
    whileConfig: {
        whileFn: ({ replChoice }) => replChoice !== choices.EXIT,
        input: replInput(scenarios),
        output: new scenarios.ScenarioOutput(
            "REPL output",
            (state) => state.replOutput,
            { preformatted: true },
        ),
    },
},
);

export { replInput, replAction, choices };

```

Vernichten Sie alle erstellten Ressourcen (clean.steps.js).

```

import {
    DeleteObjectCommand,
    DeleteBucketCommand,
    ListObjectVersionsCommand,
    GetObjectLegalHoldCommand,
    GetObjectRetentionCommand,
    PutObjectLegalHoldCommand,
} from "@aws-sdk/client-s3";

/**
 * @typedef {import("@aws-doc-sdk-examples/lib/scenario/index.js")} Scenarios
 */

/**
 * @typedef {import("@aws-sdk/client-s3").S3Client} S3Client
 */

```

```

/**
 * @param {Scenarios} scenarios
 */
const confirmCleanup = (scenarios) =>
  new scenarios.ScenarioInput("confirmCleanup", "Clean up resources?", {
    type: "confirm",
  });

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const cleanupAction = (scenarios, client) =>
  new scenarios.ScenarioAction("cleanupAction", async (state) => {
    const { noLockBucketName, lockEnabledBucketName, retentionBucketName } =
      state;

    const buckets = [
      noLockBucketName,
      lockEnabledBucketName,
      retentionBucketName,
    ];

    for (const bucket of buckets) {
      /** @type {import("@aws-sdk/client-s3").ListObjectVersionsCommandOutput} */
      let objectsResponse;

      try {
        objectsResponse = await client.send(
          new ListObjectVersionsCommand({
            Bucket: bucket,
          }),
        );
      } catch (e) {
        if (e instanceof Error && e.name === "NoSuchBucket") {
          console.log("Object's bucket has already been deleted.");
          continue;
        }
        throw e;
      }

      for (const version of objectsResponse.Versions || []) {
        const { Key, VersionId } = version;

```

```
try {
  const legalHold = await client.send(
    new GetObjectLegalHoldCommand({
      Bucket: bucket,
      Key,
      VersionId,
    }),
  );

  if (legalHold.LegalHold?.Status === "ON") {
    await client.send(
      new PutObjectLegalHoldCommand({
        Bucket: bucket,
        Key,
        VersionId,
        LegalHold: {
          Status: "OFF",
        },
      }),
    );
  }
} catch (err) {
  console.log(
    `Unable to fetch legal hold for ${Key} in ${bucket}: '${err.message}'`,
  );
}

try {
  const retention = await client.send(
    new GetObjectRetentionCommand({
      Bucket: bucket,
      Key,
      VersionId,
    }),
  );

  if (retention.Retention?.Mode === "GOVERNANCE") {
    await client.send(
      new DeleteObjectCommand({
        Bucket: bucket,
        Key,
        VersionId,
        BypassGovernanceRetention: true,
      })
    );
  }
}
```

```
        }},
      );
    }
  } catch (err) {
    console.log(
      `Unable to fetch object lock retention for ${Key} in ${bucket}:`
      `${err.message}`,
    );
  }

  await client.send(
    new DeleteObjectCommand({
      Bucket: bucket,
      Key,
      VersionId,
    }),
  );
}

await client.send(new DeleteBucketCommand({ Bucket: bucket }));
console.log(`Delete for ${bucket} complete.`);
}
});

export { confirmCleanup, cleanupAction };
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für JavaScript - API-Referenz.
 - [GetObjectLegalHold](#)
 - [GetObjectLockConfiguration](#)
 - [GetObjectRetention](#)
 - [PutObjectLegalHold](#)
 - [PutObjectLockConfiguration](#)
 - [PutObjectRetention](#)

Senden von bedingten Anforderungen

Das folgende Codebeispiel zeigt, wie Sie Vorbedingungen zu Amazon-S3-Anforderungen hinzufügen.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Einstiegspunkt für den Workflow (index.js). Dadurch werden alle Schritte orchestriert. Die Implementierungsdetails für Szenario, ScenarioInput ScenarioOutput, und finden GitHub Sie unter ScenarioAction.

```
import * as Scenarios from "@aws-doc-sdk-examples/lib/scenario/index.js";
import {
  exitOnFalse,
  loadState,
  saveState,
} from "@aws-doc-sdk-examples/lib/scenario/steps-common.js";

import { welcome, welcomeContinue } from "../welcome.steps.js";
import {
  confirmCreateBuckets,
  confirmPopulateBuckets,
  createBuckets,
  createBucketsAction,
  getBucketPrefix,
  populateBuckets,
  populateBucketsAction,
} from "../setup.steps.js";

/**
 * @param {Scenarios} scenarios
 * @param {Record<string, any>} initialState
 */
export const getWorkflowStages = (scenarios, initialState = {}) => {
  const client = new S3Client({});

  return {
    deploy: new scenarios.Scenario(
      "S3 Conditional Requests - Deploy",
      [
        welcome(scenarios),
```

```

        welcomeContinue(scenarios),
        exitOnFalse(scenarios, "welcomeContinue"),
        getBucketPrefix(scenarios),
        createBuckets(scenarios),
        confirmCreateBuckets(scenarios),
        exitOnFalse(scenarios, "confirmCreateBuckets"),
        createBucketsAction(scenarios, client),
        populateBuckets(scenarios),
        confirmPopulateBuckets(scenarios),
        exitOnFalse(scenarios, "confirmPopulateBuckets"),
        populateBucketsAction(scenarios, client),
        saveState,
    ],
    initialState,
),
demo: new scenarios.Scenario(
    "S3 Conditional Requests - Demo",
    [loadState, welcome(scenarios), replAction(scenarios, client)],
    initialState,
),
clean: new scenarios.Scenario(
    "S3 Conditional Requests - Destroy",
    [
        loadState,
        confirmCleanup(scenarios),
        exitOnFalse(scenarios, "confirmCleanup"),
        cleanupAction(scenarios, client),
    ],
    initialState,
),
};
};

// Call function if run directly
import { fileURLToPath } from "node:url";
import { S3Client } from "@aws-sdk/client-s3";
import { cleanupAction, confirmCleanup } from "./clean.steps.js";
import { replAction } from "./repl.steps.js";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
    const objectLockingScenarios = getWorkflowStages(Scenarios);
    Scenarios.parseScenarioArgs(objectLockingScenarios, {
        name: "Amazon S3 object locking workflow",
        description:

```

```

    "Work with Amazon Simple Storage Service (Amazon S3) object locking
    features.",
    synopsis:
      "node index.js --scenario <deploy | demo | clean> [-h|--help] [-y|--yes] [-v|--verbose]",
    });
  }
}

```

Geben Sie Willkommensnachrichten an die Konsole aus (welcome.steps.js).

```

/**
 * @typedef {import("@aws-doc-sdk-examples/lib/scenario/index.js")} Scenarios
 */

/**
 * @param {Scenarios} scenarios
 */
const welcome = (scenarios) =>
  new scenarios.ScenarioOutput(
    "welcome",
    "This example demonstrates the use of conditional requests for S3 operations." +
    " You can use conditional requests to add preconditions to S3 read requests to" +
    "return " +
    "or copy an object based on its Entity tag (ETag), or last modified date.You" +
    "can use " +
    "a conditional write requests to prevent overwrites by ensuring there is no" +
    "existing " +
    "object with the same key.\n" +
    "This example will enable you to perform conditional reads and writes that" +
    "will succeed " +
    "or fail based on your selected options.\n" +
    "Sample buckets and a sample object will be created as part of the example.\n" +
    "Some steps require a key name prefix to be defined by the user. Before you" +
    "begin, you can " +
    "optionally edit this prefix in ./object_name.json. If you do so, please" +
    "reload the scenario before you begin.",
    { header: true },
  );

/**
 * @param {Scenarios} scenarios

```

```

*/
const welcomeContinue = (scenarios) =>
  new scenarios.ScenarioInput(
    "welcomeContinue",
    "Press Enter when you are ready to start.",
    { type: "confirm" },
  );

export { welcome, welcomeContinue };

```

Stellen Sie Buckets und Objekte bereit (setup.steps.js).

```

import {
  ChecksumAlgorithm,
  CreateBucketCommand,
  PutObjectCommand,
  BucketAlreadyExists,
  BucketAlreadyOwnedByYou,
  S3ServiceException,
  waitUntilBucketExists,
} from "@aws-sdk/client-s3";

/**
 * @typedef {import("@aws-doc-sdk-examples/lib/scenario/index.js")} Scenarios
 */

/**
 * @typedef {import("@aws-sdk/client-s3").S3Client} S3Client
 */

/**
 * @param {Scenarios} scenarios
 */
const getBucketPrefix = (scenarios) =>
  new scenarios.ScenarioInput(
    "bucketPrefix",
    "Provide a prefix that will be used for bucket creation.",
    { type: "input", default: "amzn-s3-demo-bucket" },
  );

/**
 * @param {Scenarios} scenarios
 */

```



```

const createBuckets = (scenarios) =>
  new scenarios.ScenarioOutput(
    "createBuckets",
    (state) => `The following buckets will be created:
      ${state.bucketPrefix}-source-bucket.
      ${state.bucketPrefix}-destination-bucket.` ,
    { preformatted: true },
  );

/**
 * @param {Scenarios} scenarios
 */
const confirmCreateBuckets = (scenarios) =>
  new scenarios.ScenarioInput("confirmCreateBuckets", "Create the buckets?", {
    type: "confirm",
  });

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const createBucketsAction = (scenarios, client) =>
  new scenarios.ScenarioAction("createBucketsAction", async (state) => {
    const sourceBucketName = `${state.bucketPrefix}-source-bucket`;
    const destinationBucketName = `${state.bucketPrefix}-destination-bucket`;

    try {
      await client.send(
        new CreateBucketCommand({
          Bucket: sourceBucketName,
        }),
      );
      await waitUntilBucketExists({ client }, { Bucket: sourceBucketName });
      await client.send(
        new CreateBucketCommand({
          Bucket: destinationBucketName,
        }),
      );
      await waitUntilBucketExists(
        { client },
        { Bucket: destinationBucketName },
      );

      state.sourceBucketName = sourceBucketName;
    }
  });

```

```

        state.destinationBucketName = destinationBucketName;
    } catch (caught) {
        if (
            caught instanceof BucketAlreadyExists ||
            caught instanceof BucketAlreadyOwnedByYou
        ) {
            console.error(`${caught.name}: ${caught.message}`);
            state.earlyExit = true;
        } else {
            throw caught;
        }
    }
});

/**
 * @param {Scenarios} scenarios
 */
const populateBuckets = (scenarios) =>
    new scenarios.ScenarioOutput(
        "populateBuckets",
        (state) => `The following test files will be created:
            file01.txt in ${state.bucketPrefix}-source-bucket.`,
        { preformatted: true },
    );

/**
 * @param {Scenarios} scenarios
 */
const confirmPopulateBuckets = (scenarios) =>
    new scenarios.ScenarioInput(
        "confirmPopulateBuckets",
        "Populate the buckets?",
        { type: "confirm" },
    );

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const populateBucketsAction = (scenarios, client) =>
    new scenarios.ScenarioAction("populateBucketsAction", async (state) => {
        try {
            await client.send(
                new PutObjectCommand({

```

```
        Bucket: state.sourceBucketName,
        Key: "file01.txt",
        Body: "Content",
        ChecksumAlgorithm: ChecksumAlgorithm.SHA256,
      )),
    );
  } catch (caught) {
    if (caught instanceof S3ServiceException) {
      console.error(
        `Error from S3 while uploading object.  ${caught.name}:
${caught.message}`,
      );
    } else {
      throw caught;
    }
  }
});

export {
  confirmCreateBuckets,
  confirmPopulateBuckets,
  createBuckets,
  createBucketsAction,
  getBucketPrefix,
  populateBuckets,
  populateBucketsAction,
};
```

Rufen Sie Objekte mithilfe von bedingten S3-Anfragen (`repl.steps.js`) ab, kopieren Sie sie und fügen Sie sie ein.

```
import path from "node:path";
import { fileURLToPath } from "node:url";
import { dirname } from "node:path";

import {
  ListObjectVersionsCommand,
  GetObjectCommand,
  CopyObjectCommand,
  PutObjectCommand,
} from "@aws-sdk/client-s3";
```

```
import data from "./object_name.json" assert { type: "json" };
import { readFile } from "node:fs/promises";
import {
  ScenarioInput,
  Scenario,
  ScenarioAction,
  ScenarioOutput,
} from "../../libs/scenario/index.js";

/**
 * @typedef {import("@aws-doc-sdk-examples/lib/scenario/index.js")} Scenarios
 */

/**
 * @typedef {import("@aws-sdk/client-s3").S3Client} S3Client
 */

const choices = {
  EXIT: 0,
  LIST_ALL_FILES: 1,
  CONDITIONAL_READ: 2,
  CONDITIONAL_COPY: 3,
  CONDITIONAL_WRITE: 4,
};

/**
 * @param {Scenarios} scenarios
 */
const replInput = (scenarios) =>
  new ScenarioInput(
    "replChoice",
    "Explore the S3 conditional request features by selecting one of the following choices",
    {
      type: "select",
      choices: [
        { name: "Print list of bucket items.", value: choices.LIST_ALL_FILES },
        {
          name: "Perform a conditional read.",
          value: choices.CONDITIONAL_READ,
        },
        {
          name: "Perform a conditional copy. These examples use the key name prefix defined in ./object_name.json.",

```

```

        value: choices.CONDITIONAL_COPY,
      },
      {
        name: "Perform a conditional write. This example use the sample file ./
text02.txt.",
        value: choices.CONDITIONAL_WRITE,
      },
      { name: "Finish the workflow.", value: choices.EXIT },
    ],
  },
);

/**
 * @param {S3Client} client
 * @param {string[]} buckets
 */
const getAllFiles = async (client, buckets) => {
  /** @type {{bucket: string, key: string, version: string}[]} */
  const files = [];
  for (const bucket of buckets) {
    const objectsResponse = await client.send(
      new ListObjectVersionsCommand({ Bucket: bucket }),
    );
    for (const version of objectsResponse.Versions || []) {
      const { Key } = version;
      files.push({ bucket, key: Key });
    }
  }
  return files;
};

/**
 * @param {S3Client} client
 * @param {string[]} buckets
 * @param {string} key
 */
const getEtag = async (client, bucket, key) => {
  const objectsResponse = await client.send(
    new GetObjectCommand({
      Bucket: bucket,
      Key: key,
    }),
  );
  return objectsResponse.ETag;
};

```

```

};

/**
 * @param {S3Client} client
 * @param {string[]} buckets
 */

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
export const replAction = (scenarios, client) =>
  new ScenarioAction(
    "replAction",
    async (state) => {
      const files = await getAllFiles(client, [
        state.sourceBucketName,
        state.destinationBucketName,
      ]);

      const fileInput = new scenarios.ScenarioInput(
        "selectedFile",
        "Select a file to use:",
        {
          type: "select",
          choices: files.map((file, index) => ({
            name: `${index + 1}: ${file.bucket}: ${file.key} (Etag: ${
              file.version
            })`,
            value: index,
          })),
        },
      );

      const condReadOptions = new scenarios.ScenarioInput(
        "selectOption",
        "Which conditional read action would you like to take?",
        {
          type: "select",
          choices: [
            "If-Match: using the object's ETag. This condition should succeed.",
            "If-None-Match: using the object's ETag. This condition should fail.",
            "If-Modified-Since: using yesterday's date. This condition should
succeed.",

```

```

        "If-Unmodified-Since: using yesterday's date. This condition should
fail.",
    ],
    },
);
const condCopyOptions = new scenarios.ScenarioInput(
    "selectOption",
    "Which conditional copy action would you like to take?",
    {
        type: "select",
        choices: [
            "If-Match: using the object's ETag. This condition should succeed.",
            "If-None-Match: using the object's ETag. This condition should fail.",
            "If-Modified-Since: using yesterday's date. This condition should
succeed.",
            "If-Unmodified-Since: using yesterday's date. This condition should
fail.",
        ],
    },
);
const condWriteOptions = new scenarios.ScenarioInput(
    "selectOption",
    "Which conditional write action would you like to take?",
    {
        type: "select",
        choices: [
            "IfNoneMatch condition on the object key: If the key is a duplicate, the
write will fail.",
        ],
    },
);

const { replChoice } = state;

switch (replChoice) {
    case choices.LIST_ALL_FILES: {
        const files = await getAllFiles(client, [
            state.sourceBucketName,
            state.destinationBucketName,
        ]);
        state.replOutput = files
            .map(
                (file) => `Items in bucket ${file.bucket}: object: ${file.key} `,
            )
    }
}

```

```

        .join("\n");
    break;
}
case choices.CONDITIONAL_READ:
{
    const selectedCondRead = await condReadOptions.handle(state);
    if (
        selectedCondRead ===
        "If-Match: using the object's ETag. This condition should succeed."
    ) {
        const bucket = state.sourceBucketName;
        const key = "file01.txt";
        const ETag = await getEtag(client, bucket, key);

        try {
            await client.send(
                new GetObjectCommand({
                    Bucket: bucket,
                    Key: key,
                    IfMatch: ETag,
                }),
            );
            state.replOutput = `${key} in bucket ${state.sourceBucketName} read
because ETag provided matches the object's ETag.`;
        } catch (err) {
            state.replOutput = `Unable to read object ${key} in bucket
${state.sourceBucketName}: ${err.message}`;
        }
        break;
    }
    if (
        selectedCondRead ===
        "If-None-Match: using the object's ETag. This condition should fail."
    ) {
        const bucket = state.sourceBucketName;
        const key = "file01.txt";
        const ETag = await getEtag(client, bucket, key);

        try {
            await client.send(
                new GetObjectCommand({
                    Bucket: bucket,
                    Key: key,
                    IfNoneMatch: ETag,

```



```

        })),
    );
    state.replOutput = `${key} in ${state.sourceBucketName} was
returned.`;
    } catch (err) {
        state.replOutput = `${key} in ${state.sourceBucketName} was not
read: ${err.message}`;
    }
    break;
}
if (
    selectedCondRead ===
    "If-Modified-Since: using yesterday's date. This condition should
succeed."
) {
    const date = new Date();
    date.setDate(date.getDate() - 1);

    const bucket = state.sourceBucketName;
    const key = "file01.txt";
    try {
        await client.send(
            new GetObjectCommand({
                Bucket: bucket,
                Key: key,
                IfModifiedSince: date,
            }),
        );
        state.replOutput = `${key} in bucket ${state.sourceBucketName} read
because it has been created or modified in the last 24 hours.`;
    } catch (err) {
        state.replOutput = `Unable to read object ${key} in bucket
${state.sourceBucketName}: ${err.message}`;
    }
    break;
}
if (
    selectedCondRead ===
    "If-Unmodified-Since: using yesterday's date. This condition should
fail."
) {
    const bucket = state.sourceBucketName;
    const key = "file01.txt";

```

```

    const date = new Date();
    date.setDate(date.getDate() - 1);
    try {
      await client.send(
        new GetObjectCommand({
          Bucket: bucket,
          Key: key,
          IfUnmodifiedSince: date,
        }),
      );
      state replOutput = `${key} in ${state.sourceBucketName} was read.`;
    } catch (err) {
      state replOutput = `${key} in ${state.sourceBucketName} was not
read: ${err.message}`;
    }
    break;
  }
}
break;
case choices.CONDITIONAL_COPY: {
  const selectedCondCopy = await condCopyOptions.handle(state);
  if (
    selectedCondCopy ===
    "If-Match: using the object's ETag. This condition should succeed."
  ) {
    const bucket = state.sourceBucketName;
    const key = "file01.txt";
    const ETag = await getEtag(client, bucket, key);

    const copySource = `${bucket}/${key}`;
    // Optionally edit the default key name prefix of the copied object
in ./object_name.json.
    const name = data.name;
    const copiedKey = `${name}${key}`;
    try {
      await client.send(
        new CopyObjectCommand({
          CopySource: copySource,
          Bucket: state.destinationBucketName,
          Key: copiedKey,
          CopySourceIfMatch: ETag,
        }),
      );
    }
  }
}

```

```

        state.replOutput = `${key} copied as ${copiedKey} to bucket
${state.destinationBucketName} because ETag provided matches the object's ETag.`;
    } catch (err) {
        state.replOutput = `Unable to copy object ${key} as ${copiedKey} to
bucket ${state.destinationBucketName}: ${err.message}`;
    }
    break;
}
if (
    selectedCondCopy ===
    "If-None-Match: using the object's ETag. This condition should fail."
) {
    const bucket = state.sourceBucketName;
    const key = "file01.txt";
    const ETag = await getEtag(client, bucket, key);
    const copySource = `${bucket}/${key}`;
    // Optionally edit the default key name prefix of the copied object
in ./object_name.json.
    const name = data.name;
    const copiedKey = `${name}${key}`;

    try {
        await client.send(
            new CopyObjectCommand({
                CopySource: copySource,
                Bucket: state.destinationBucketName,
                Key: copiedKey,
                CopySourceIfNoneMatch: ETag,
            }),
        );
        state.replOutput = `${copiedKey} copied to bucket
${state.destinationBucketName}`;
    } catch (err) {
        state.replOutput = `Unable to copy object as ${key} as as ${copiedKey}
to bucket ${state.destinationBucketName}: ${err.message}`;
    }
    break;
}
if (
    selectedCondCopy ===
    "If-Modified-Since: using yesterday's date. This condition should
succeed."
) {
    const bucket = state.sourceBucketName;

```

```

    const key = "file01.txt";
    const copySource = `${bucket}/${key}`;
    // Optionally edit the default key name prefix of the copied object
in ./object_name.json.
    const name = data.name;
    const copiedKey = `${name}${key}`;

    const date = new Date();
    date.setDate(date.getDate() - 1);

    try {
      await client.send(
        new CopyObjectCommand({
          CopySource: copySource,
          Bucket: state.destinationBucketName,
          Key: copiedKey,
          CopySourceIfModifiedSince: date,
        }),
      );
      state.replOutput = `${key} copied as ${copiedKey} to bucket
${state.destinationBucketName} because it has been created or modified in the last
24 hours.`;
    } catch (err) {
      state.replOutput = `Unable to copy object ${key} as ${copiedKey} to
bucket ${state.destinationBucketName} : ${err.message}`;
    }
    break;
  }
  if (
    selectedCondCopy ===
    "If-Unmodified-Since: using yesterday's date. This condition should
fail."
  ) {
    const bucket = state.sourceBucketName;
    const key = "file01.txt";
    const copySource = `${bucket}/${key}`;
    // Optionally edit the default key name prefix of the copied object
in ./object_name.json.
    const name = data.name;
    const copiedKey = `${name}${key}`;

    const date = new Date();
    date.setDate(date.getDate() - 1);

```

```

    try {
      await client.send(
        new CopyObjectCommand({
          CopySource: copySource,
          Bucket: state.destinationBucketName,
          Key: copiedKey,
          CopySourceIfUnmodifiedSince: date,
        }),
      );
      state.replOutput = `${copiedKey} copied to bucket
${state.destinationBucketName} because it has not been created or modified in the
last 24 hours.`;
    } catch (err) {
      state.replOutput = `Unable to copy object ${key} to bucket
${state.destinationBucketName}: ${err.message}`;
    }
  }
  break;
}
case choices.CONDITIONAL_WRITE:
{
  const selectedCondWrite = await condWriteOptions.handle(state);
  if (
    selectedCondWrite ===
    "IfNoneMatch condition on the object key: If the key is a duplicate,
the write will fail."
  ) {
    // Optionally edit the default key name prefix of the copied object
    in ./object_name.json.
    const key = "text02.txt";
    const __filename = fileURLToPath(import.meta.url);
    const __dirname = dirname(__filename);
    const filePath = path.join(__dirname, "text02.txt");
    try {
      await client.send(
        new PutObjectCommand({
          Bucket: `${state.destinationBucketName}`,
          Key: `${key}`,
          Body: await readFile(filePath),
          IfNoneMatch: "*",
        }),
      );
      state.replOutput = `${key} uploaded to bucket
${state.destinationBucketName} because the key is not a duplicate.`;
    }
  }
}

```

```

        } catch (err) {
            state.replOutput = `Unable to upload object to bucket
${state.destinationBucketName}:${err.message}`;
        }
        break;
    }
}
break;

default:
    throw new Error(`Invalid replChoice: ${replChoice}`);
},
{
    whileConfig: {
        whileFn: ({ replChoice }) => replChoice !== choices.EXIT,
        input: replInput(scenarios),
        output: new ScenarioOutput("REPL output", (state) => state.replOutput, {
            preformatted: true,
        }),
    },
},
);

export { replInput, choices };

```

Vernichten Sie alle erstellten Ressourcen (clean.steps.js).

```

import {
    DeleteObjectCommand,
    DeleteBucketCommand,
    ListObjectVersionsCommand,
} from "@aws-sdk/client-s3";

/**
 * @typedef {import("@aws-doc-sdk-examples/lib/scenario/index.js")} Scenarios
 */

/**
 * @typedef {import("@aws-sdk/client-s3").S3Client} S3Client
 */

```

```
/**
 * @param {Scenarios} scenarios
 */
const confirmCleanup = (scenarios) =>
  new scenarios.ScenarioInput("confirmCleanup", "Clean up resources?", {
    type: "confirm",
  });

/**
 * @param {Scenarios} scenarios
 * @param {S3Client} client
 */
const cleanupAction = (scenarios, client) =>
  new scenarios.ScenarioAction("cleanupAction", async (state) => {
    const { sourceBucketName, destinationBucketName } = state;
    const buckets = [sourceBucketName, destinationBucketName].filter((b) => b);

    for (const bucket of buckets) {
      try {
        let objectsResponse;
        objectsResponse = await client.send(
          new ListObjectVersionsCommand({
            Bucket: bucket,
          }),
        );
        for (const version of objectsResponse.Versions || []) {
          const { Key, VersionId } = version;
          try {
            await client.send(
              new DeleteObjectCommand({
                Bucket: bucket,
                Key,
                VersionId,
              }),
            );
          } catch (err) {
            console.log(`An error occurred: ${err.message} `);
          }
        }
      } catch (e) {
        if (e instanceof Error && e.name === "NoSuchBucket") {
          console.log("Objects and buckets have already been deleted.");
          continue;
        }
      }
    }
  })
```

```
        throw e;
    }

    await client.send(new DeleteBucketCommand({ Bucket: bucket }));
    console.log(`Delete for ${bucket} complete.`);
  }
});

export { confirmCleanup, cleanupAction };
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für JavaScript - API-Referenz.
 - [CopyObject](#)
 - [GetObject](#)
 - [PutObject](#)

Hoch- oder Herunterladen großer Dateien

Das folgende Codebeispiel veranschaulicht, wie Sie große Dateien zu S3 hochladen bzw. von S3 herunterladen.

Weitere Informationen finden Sie unter [Hochladen eines Objekts mit Multipart-Upload](#).

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Laden Sie eine große Datei hoch.

```
import { S3Client } from "@aws-sdk/client-s3";
import { Upload } from "@aws-sdk/lib-storage";

import {
  ProgressBar,
  logger,
```



```
} from "@aws-doc-sdk-examples/lib/utils/util-log.js";

const twentyFiveMB = 25 * 1024 * 1024;

export const createString = (size = twentyFiveMB) => {
  return "x".repeat(size);
};

/**
 * Create a 25MB file and upload it in parts to the specified
 * Amazon S3 bucket.
 * @param {{ bucketName: string, key: string }}
 */
export const main = async ({ bucketName, key }) => {
  const str = createString();
  const buffer = Buffer.from(str, "utf8");
  const progressBar = new ProgressBar({
    description: `Uploading "${key}" to "${bucketName}"`,
    barLength: 30,
  });

  try {
    const upload = new Upload({
      client: new S3Client({}),
      params: {
        Bucket: bucketName,
        Key: key,
        Body: buffer,
      },
    });

    upload.on("httpUploadProgress", ({ loaded, total }) => {
      progressBar.update({ current: loaded, total });
    });

    await upload.done();
  } catch (caught) {
    if (caught instanceof Error && caught.name === "AbortError") {
      logger.error(`Multipart upload was aborted. ${caught.message}`);
    } else {
      throw caught;
    }
  }
};
```

Laden Sie eine große Datei herunter.

```
import { fileURLToPath } from "node:url";
import { GetObjectCommand, NoSuchKey, S3Client } from "@aws-sdk/client-s3";
import { createWriteStream, rmSync } from "node:fs";

const s3Client = new S3Client({});
const oneMB = 1024 * 1024;

export const getObjectRange = ({ bucket, key, start, end }) => {
  const command = new GetObjectCommand({
    Bucket: bucket,
    Key: key,
    Range: `bytes=${start}-${end}`,
  });

  return s3Client.send(command);
};

/**
 * @param {string | undefined} contentRange
 */
export const getRangeAndLength = (contentRange) => {
  const [range, length] = contentRange.split("/");
  const [start, end] = range.split("-");
  return {
    start: Number.parseInt(start),
    end: Number.parseInt(end),
    length: Number.parseInt(length),
  };
};

export const isComplete = ({ end, length }) => end === length - 1;

const downloadInChunks = async ({ bucket, key }) => {
  const writeStream = createWriteStream(
    fileURLToPath(new URL(`./${key}`, import.meta.url)),
  ).on("error", (err) => console.error(err));

  let rangeAndLength = { start: -1, end: -1, length: -1 };
```

```

while (!isComplete(rangeAndLength)) {
  const { end } = rangeAndLength;
  const nextRange = { start: end + 1, end: end + oneMB };

  const { ContentRange, Body } = await getObjectRange({
    bucket,
    key,
    ...nextRange,
  });
  console.log(`Downloaded bytes ${nextRange.start} to ${nextRange.end}`);

  writeStream.write(await Body.transformToByteArray());
  rangeAndLength = getRangeAndLength(ContentRange);
}
};

/**
 * Download a large object from and Amazon S3 bucket.
 *
 * When downloading a large file, you might want to break it down into
 * smaller pieces. Amazon S3 accepts a Range header to specify the start
 * and end of the byte range to be downloaded.
 *
 * @param {{ bucketName: string, key: string }}
 */
export const main = async ({ bucketName, key }) => {
  try {
    await downloadInChunks({
      bucket: bucketName,
      key: key,
    });
  } catch (caught) {
    if (caught instanceof NoSuchKey) {
      console.error(`Failed to download object. No such key "${key}".`);
      rmSync(key);
    }
  }
};

```

Serverless-Beispiele

Aufrufen einer Lambda-Funktion über einen Amazon-S3-Auslöser

Im folgenden Codebeispiel wird die Implementierung einer Lambda-Funktion gezeigt, die ein Ereignis empfängt, das durch Hochladen eines Objekts in einen S3-Bucket ausgelöst wird. Die Funktion ruft den Namen des S3-Buckets sowie den Objektschlüssel aus dem Ereignisparameter ab und ruft die Amazon-S3-API auf, um den Inhaltstyp des Objekts abzurufen und zu protokollieren.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Konsumieren eines S3-Ereignisses mit Lambda unter Verwendung JavaScript.

```
import { S3Client, HeadObjectCommand } from "@aws-sdk/client-s3";

const client = new S3Client();

export const handler = async (event, context) => {

  // Get the object from the event and show its content type
  const bucket = event.Records[0].s3.bucket.name;
  const key = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, ' '));

  try {
    const { ContentType } = await client.send(new HeadObjectCommand({
      Bucket: bucket,
      Key: key,
    }));

    console.log('CONTENT TYPE:', ContentType);
    return ContentType;
  } catch (err) {
    console.log(err);
  }
}
```

```
    const message = `Error getting object ${key} from bucket ${bucket}. Make
sure they exist and your bucket is in the same region as this function.`;
    console.log(message);
    throw new Error(message);
  }
};
```

Konsumieren eines S3-Ereignisses mit Lambda unter Verwendung TypeScript.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { S3Event } from 'aws-lambda';
import { S3Client, HeadObjectCommand } from '@aws-sdk/client-s3';

const s3 = new S3Client({ region: process.env.AWS_REGION });

export const handler = async (event: S3Event): Promise<string | undefined> => {
  // Get the object from the event and show its content type
  const bucket = event.Records[0].s3.bucket.name;
  const key = decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, '
'));
  const params = {
    Bucket: bucket,
    Key: key,
  };
  try {
    const { ContentType } = await s3.send(new HeadObjectCommand(params));
    console.log('CONTENT TYPE:', ContentType);
    return ContentType;
  } catch (err) {
    console.log(err);
    const message = `Error getting object ${key} from bucket ${bucket}. Make sure
they exist and your bucket is in the same region as this function.`;
    console.log(message);
    throw new Error(message);
  }
};
```

SageMaker KI-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK für JavaScript (v3) mit SageMaker KI Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien anzeigen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb eines Service aufrufen oder mit anderen AWS-Services kombinieren.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kodex finden.

Themen

- [Erste Schritte](#)
- [Aktionen](#)
- [Szenarien](#)

Erste Schritte

Hallo SageMaker AI

Das folgende Codebeispiel zeigt, wie Sie mit SageMaker KI beginnen können.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import {
  SageMakerClient,
  ListNotebookInstancesCommand,
```

```
} from "@aws-sdk/client-sagemaker";

const client = new SageMakerClient({
  region: "us-west-2",
});

export const helloSagemaker = async () => {
  const command = new ListNotebookInstancesCommand({ MaxResults: 5 });

  const response = await client.send(command);
  console.log(
    "Hello Amazon SageMaker! Let's list some of your notebook instances:",
  );

  const instances = response.NotebookInstances || [];

  if (instances.length === 0) {
    console.log(
      "• No notebook instances found. Try creating one in the AWS Management Console  
or with the CreateNotebookInstanceCommand.",
    );
  } else {
    console.log(
      instances
        .map(
          (i) =>
            `• Instance: ${i.NotebookInstanceName}\n  Arn:${i.NotebookInstanceArn}\n  Creation Date: ${i.CreationTime.toISOString()}`,
        )
        .join("\n"),
    );
  }

  return response;
};
```

- Einzelheiten zur API finden Sie [ListNotebookInstances](#) in der AWS SDK für JavaScript API-Referenz.

Aktionen

CreatePipeline

Das folgende Codebeispiel zeigt die Verwendung `CreatePipeline`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Eine Funktion, die mithilfe einer lokal bereitgestellten JSON-Definition eine SageMaker KI-Pipeline erstellt.

```
/**
 * Create the Amazon SageMaker pipeline using a JSON pipeline definition. The
 * definition
 * can also be provided as an Amazon S3 object using PipelineDefinitionS3Location.
 * @param {{roleArn: string, name: string, sagemakerClient: import('@aws-sdk/client-
 * sagemaker').SageMakerClient}} props
 */
export async function createSagemakerPipeline({
  // Assumes an AWS IAM role has been created for this pipeline.
  roleArn,
  name,
  // Assumes an AWS Lambda function has been created for this pipeline.
  functionArn,
  sagemakerClient,
}) {
  const pipelineDefinition = readFileSync(
    // dirnameFromMetaUrl is a local utility function. You can find its
    implementation
    // on GitHub.
    `${dirnameFromMetaUrl(
      import.meta.url,
    )}../../../../scenarios/features/sagemaker_pipelines/resources/
    GeoSpatialPipeline.json`,
  )
  .toString()
```



```
.replace(/\\*FUNCTION_ARN\\*/g, functionArn);

let arn = null;

const createPipeline = () =>
  sagemakerClient.send(
    new CreatePipelineCommand({
      PipelineName: name,
      PipelineDefinition: pipelineDefinition,
      RoleArn: roleArn,
    }),
  );

try {
  const { PipelineArn } = await createPipeline();
  arn = PipelineArn;
} catch (caught) {
  if (
    caught instanceof Error &&
    caught.name === "ValidationException" &&
    caught.message.includes(
      "Pipeline names must be unique within an AWS account and region",
    )
  ) {
    const { PipelineArn } = await sagemakerClient.send(
      new DescribePipelineCommand({ PipelineName: name }),
    );
    arn = PipelineArn;
  } else {
    throw caught;
  }
}

return {
  arn,
  cleanUp: async () => {
    await sagemakerClient.send(
      new DeletePipelineCommand({ PipelineName: name }),
    );
  },
};
}
```

- Einzelheiten zur API finden Sie [CreatePipeline](#) unter AWS SDK für JavaScript API-Referenz.

DeletePipeline

Das folgende Codebeispiel zeigt die Verwendung `DeletePipeline`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Die Syntax zum Löschen einer SageMaker AI-Pipeline. Dieser Code ist Teil einer größeren Funktion. Weitere Informationen finden Sie unter „Eine Pipeline erstellen“ oder im GitHub Repository.

```
await sagemakerClient.send(  
    new DeletePipelineCommand({ PipelineName: name } ),  
);
```

- Einzelheiten zur API finden Sie unter [DeletePipeline](#) API-Referenz. AWS SDK für JavaScript

DescribePipelineExecution

Das folgende Codebeispiel zeigt die Verwendung `DescribePipelineExecution`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Warten Sie, bis die Ausführung einer SageMaker KI-Pipeline erfolgreich ist, fehlschlägt oder beendet wird.

```

/**
 * Poll the executing pipeline until the status is 'SUCCEEDED', 'STOPPED', or
 * 'FAILED'.
 * @param {{ arn: string, sagemakerClient: import('@aws-sdk/client-
 * sagemaker').SageMakerClient, wait: (ms: number) => Promise<void>}} props
 */
export async function waitForPipelineComplete({ arn, sagemakerClient, wait }) {
  const command = new DescribePipelineExecutionCommand({
    PipelineExecutionArn: arn,
  });

  let complete = false;
  const intervalInSeconds = 15;
  const COMPLETION_STATUSES = [
    PipelineExecutionStatus.FAILED,
    PipelineExecutionStatus.STOPPED,
    PipelineExecutionStatus.SUCCEEDED,
  ];

  do {
    const { PipelineExecutionStatus: status, FailureReason } =
      await sagemakerClient.send(command);

    complete = COMPLETION_STATUSES.includes(status);

    if (!complete) {
      console.log(
        `Pipeline is ${status}. Waiting ${intervalInSeconds} seconds before checking
        again.`
      );
      await wait(intervalInSeconds);
    } else if (status === PipelineExecutionStatus.FAILED) {
      throw new Error(`Pipeline failed because: ${FailureReason}`);
    } else if (status === PipelineExecutionStatus.STOPPED) {
      throw new Error("Pipeline was forcefully stopped.");
    } else {
      console.log(`Pipeline execution ${status}.`);
    }
  } while (!complete);
}

```

- Einzelheiten zur API finden Sie [DescribePipelineExecution](#) unter AWS SDK für JavaScript API-Referenz.

StartPipelineExecution

Das folgende Codebeispiel zeigt die Verwendung `StartPipelineExecution`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Starten Sie die Ausführung einer SageMaker KI-Pipeline.

```
/**
 * Start the execution of the Amazon SageMaker pipeline. Parameters that are
 * passed in are used in the AWS Lambda function.
 * @param {{
 *   name: string,
 *   sagemakerClient: import('@aws-sdk/client-sagemaker').SageMakerClient,
 *   roleArn: string,
 *   queueUrl: string,
 *   s3InputBucketName: string,
 * }} props
 */
export async function startPipelineExecution({
  sagemakerClient,
  name,
  bucketName,
  roleArn,
  queueUrl,
}) {
  /**
   * The Vector Enrichment Job requests CSV data. This configuration points to a CSV
   * file in an Amazon S3 bucket.
   * @type {import("@aws-sdk/client-sagemaker-geospatial").VectorEnrichmentJobInputConfig}
   */
  const inputConfig = {
```

```

DataSourceConfig: {
  S3Data: {
    S3Uri: `s3://${bucketName}/input/sample_data.csv`,
  },
},
DocumentType: VectorEnrichmentJobDocumentType.CSV,
};

/**
 * The Vector Enrichment Job adds additional data to the source CSV. This
configuration points
 * to an Amazon S3 prefix where the output will be stored.
 * @type {import("@aws-sdk/client-sagemaker-
geospatial").ExportVectorEnrichmentJobOutputConfig}
 */
const outputConfig = {
  S3Data: {
    S3Uri: `s3://${bucketName}/output/`,
  },
};

/**
 * This job will be a Reverse Geocoding Vector Enrichment Job. Reverse Geocoding
requires
 * latitude and longitude values.
 * @type {import("@aws-sdk/client-sagemaker-
geospatial").VectorEnrichmentJobConfig}
 */
const jobConfig = {
  ReverseGeocodingConfig: {
    XAttributeName: "Longitude",
    YAttributeName: "Latitude",
  },
};

const { PipelineExecutionArn } = await sagemakerClient.send(
  new StartPipelineExecutionCommand({
    PipelineName: name,
    PipelineExecutionDisplayName: `${name}-example-execution`,
    PipelineParameters: [
      { Name: "parameter_execution_role", Value: roleArn },
      { Name: "parameter_queue_url", Value: queueUrl },
      {
        Name: "parameter_vej_input_config",

```

```
        Value: JSON.stringify(inputConfig),
      },
      {
        Name: "parameter_vej_export_config",
        Value: JSON.stringify(outputConfig),
      },
      {
        Name: "parameter_step_1_vej_config",
        Value: JSON.stringify(jobConfig),
      },
    ],
  )),
);

return {
  arn: PipelineExecutionArn,
};
}
```

- Einzelheiten zur API finden Sie [StartPipelineExecution](#) unter AWS SDK für JavaScript API-Referenz.

Szenarien

Erste Schritte mit Geodatenaufträgen und Pipelines

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Richten Sie Ressourcen für eine Pipeline ein.
- Richten Sie eine Pipeline ein, die einen Geodatenauftrag ausführt.
- Starten Sie eine Pipeline-Ausführung.
- Überwachen Sie den Status der Ausführung.
- Zeigen Sie die Ausgabe der Pipeline an.
- Bereinigen Sie die Ressourcen.

Weitere Informationen finden Sie unter [SageMaker Pipelines mithilfe AWS SDKs von Community.aws erstellen und ausführen](#).

SDK für (v3) JavaScript

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Der folgende Dateiauszug enthält Funktionen, die den SageMaker AI-Client zur Verwaltung einer Pipeline verwenden.

```
import { readFileSync } from "node:fs";

import {
  CreateRoleCommand,
  DeleteRoleCommand,
  CreatePolicyCommand,
  DeletePolicyCommand,
  AttachRolePolicyCommand,
  DetachRolePolicyCommand,
  GetRoleCommand,
  ListPoliciesCommand,
} from "@aws-sdk/client-iam";

import {
  PublishLayerVersionCommand,
  DeleteLayerVersionCommand,
  CreateFunctionCommand,
  Runtime,
  DeleteFunctionCommand,
  CreateEventSourceMappingCommand,
  DeleteEventSourceMappingCommand,
  GetFunctionCommand,
} from "@aws-sdk/client-lambda";

import {
  PutObjectCommand,
  CreateBucketCommand,
  DeleteBucketCommand,
  DeleteObjectCommand,
  GetObjectCommand,
  ListObjectsV2Command,
```

```
} from "@aws-sdk/client-s3";

import {
  CreatePipelineCommand,
  DeletePipelineCommand,
  DescribePipelineCommand,
  DescribePipelineExecutionCommand,
  PipelineExecutionStatus,
  StartPipelineExecutionCommand,
} from "@aws-sdk/client-sagemaker";

import { VectorEnrichmentJobDocumentType } from "@aws-sdk/client-sagemaker-geospatial";

import {
  CreateQueueCommand,
  DeleteQueueCommand,
  GetQueueAttributesCommand,
  GetQueueUrlCommand,
} from "@aws-sdk/client-sqs";

import { dirnameFromMetaUrl } from "@aws-doc-sdk-examples/lib/utils/util-fs.js";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

/**
 * Create the AWS IAM role that will be assumed by AWS Lambda.
 * @param {{ name: string, iamClient: import('@aws-sdk/client-iam').IAMClient }}
 * props
 */
export async function createLambdaExecutionRole({ name, iamClient }) {
  const createRole = () =>
    iamClient.send(
      new CreateRoleCommand({
        RoleName: name,
        AssumeRolePolicyDocument: JSON.stringify({
          Version: "2012-10-17",
          Statement: [
            {
              Effect: "Allow",
              Action: ["sts:AssumeRole"],
              Principal: { Service: ["lambda.amazonaws.com"] },
            },
          ],
        }),
      })
    );
}
```



```

    }),
  );

let role = null;

try {
  const { Role } = await createRole();
  role = Role;
} catch (caught) {
  if (
    caught instanceof Error &&
    caught.name === "EntityAlreadyExistsException"
  ) {
    const { Role } = await iamClient.send(
      new GetRoleCommand({ RoleName: name }),
    );
    role = Role;
  } else {
    throw caught;
  }
}

return {
  arn: role.Arn,
  cleanUp: async () => {
    await iamClient.send(new DeleteRoleCommand({ RoleName: name }));
  },
};
}

/**
 * Create an AWS IAM policy that will be attached to the AWS IAM role assumed by the
 * AWS Lambda function.
 * The policy grants permission to work with Amazon SQS, Amazon CloudWatch, and
 * Amazon SageMaker.
 * @param {{name: string, iamClient: import('@aws-sdk/client-iam').IAMClient,
 * pipelineExecutionRoleArn: string}} props
 */
export async function createLambdaExecutionPolicy({
  name,
  iamClient,
  pipelineExecutionRoleArn,
}) {
  const policyConfig = {

```

```

Version: "2012-10-17",
Statement: [
  {
    Effect: "Allow",
    Action: [
      "sqs:ReceiveMessage",
      "sqs:DeleteMessage",
      "sqs:GetQueueAttributes",
      "logs:CreateLogGroup",
      "logs:CreateLogStream",
      "logs:PutLogEvents",
      "sagemaker-geospatial:StartVectorEnrichmentJob",
      "sagemaker-geospatial:GetVectorEnrichmentJob",
      "sagemaker:SendPipelineExecutionStepFailure",
      "sagemaker:SendPipelineExecutionStepSuccess",
      "sagemaker-geospatial:ExportVectorEnrichmentJob",
    ],
    Resource: "*",
  },
  {
    Effect: "Allow",
    // The AWS Lambda function needs permission to pass the pipeline execution
    role to
    // the StartVectorEnrichmentCommand. This restriction prevents an AWS Lambda
    function
    // from elevating privileges. For more information, see:
    // https://docs.aws.amazon.com/IAM/latest/UserGuide/
    id_roles_use_passrole.html
    Action: ["iam:PassRole"],
    Resource: `${pipelineExecutionRoleArn}`,
    Condition: {
      StringEquals: {
        "iam:PassedToService": [
          "sagemaker.amazonaws.com",
          "sagemaker-geospatial.amazonaws.com",
        ],
      },
    },
  },
],
};

const createPolicy = () =>
  iamClient.send(

```

```

    new CreatePolicyCommand({
      PolicyDocument: JSON.stringify(policyConfig),
      PolicyName: name,
    }),
  );

let policy = null;

try {
  const { Policy } = await createPolicy();
  policy = Policy;
} catch (caught) {
  if (
    caught instanceof Error &&
    caught.name === "EntityAlreadyExistsException"
  ) {
    const { Policies } = await iamClient.send(new ListPoliciesCommand({}));
    if (Policies) {
      policy = Policies.find((p) => p.PolicyName === name);
    } else {
      throw new Error("No policies found.");
    }
  } else {
    throw caught;
  }
}

return {
  arn: policy?.Arn,
  policyConfig,
  cleanUp: async () => {
    await iamClient.send(new DeletePolicyCommand({ PolicyArn: policy?.Arn }));
  },
};
}

/**
 * Attach an AWS IAM policy to an AWS IAM role.
 * @param {{roleName: string, policyArn: string, iamClient: import('@aws-sdk/client-iam').IAMClient}} props
 */
export async function attachPolicy({ roleName, policyArn, iamClient }) {
  const attachPolicyCommand = new AttachRolePolicyCommand({
    RoleName: roleName,
  });

```

```

    PolicyArn: policyArn,
  });

  await iamClient.send(attachPolicyCommand);
  return {
    cleanUp: async () => {
      await iamClient.send(
        new DetachRolePolicyCommand({
          RoleName: roleName,
          PolicyArn: policyArn,
        }),
      );
    },
  };
}

/**
 * Create an AWS Lambda layer that contains the Amazon SageMaker and Amazon
 * SageMaker Geospatial clients
 * in the runtime. The default runtime supports v3.188.0 of the JavaScript SDK. The
 * Amazon SageMaker
 * Geospatial client wasn't introduced until v3.221.0.
 * @param {{ name: string, lambdaClient: import('@aws-sdk/client-
 * lambda').LambdaClient }} props
 */
export async function createLambdaLayer({ name, lambdaClient }) {
  const layerPath = `${dirnameFromMetaUrl(import.meta.url)}lambda/nodejs.zip`;
  const { LayerVersionArn, Version } = await lambdaClient.send(
    new PublishLayerVersionCommand({
      LayerName: name,
      Content: {
        ZipFile: Uint8Array.from(readFileSync(layerPath)),
      },
    }),
  );

  return {
    versionArn: LayerVersionArn,
    version: Version,
    cleanUp: async () => {
      await lambdaClient.send(
        new DeleteLayerVersionCommand({
          LayerName: name,
          VersionNumber: Version,

```

```

    }},
  );
},
};
}

/**
 * Deploy the AWS Lambda function that will be used to respond to Amazon SageMaker
 * pipeline
 * execution steps.
 * @param {{roleArn: string, name: string, lambdaClient: import('@aws-sdk/client-
 * lambda').LambdaClient, layerVersionArn: string}} props
 */
export async function createLambdaFunction({
  name,
  roleArn,
  lambdaClient,
  layerVersionArn,
}) {
  const lambdaPath = `${dirnameFromMetaUrl(
    import.meta.url,
  )}lambda/dist/index.mjs.zip`;

  // If a function of the same name already exists, return that
  // function's ARN instead. By default this is
  // "sagemaker-wkflw-lambda-function", so collisions are
  // unlikely.
  const createFunction = async () => {
    try {
      return await lambdaClient.send(
        new CreateFunctionCommand({
          Code: {
            ZipFile: Uint8Array.from(readFileSync(lambdaPath)),
          },
          Runtime: Runtime.nodejs18x,
          Handler: "index.handler",
          Layers: [layerVersionArn],
          FunctionName: name,
          Role: roleArn,
        })),
    );
  } catch (caught) {
    if (
      caught instanceof Error &&

```

```

        caught.name === "ResourceConflictException"
    ) {
        const { Configuration } = await lambdaClient.send(
            new GetFunctionCommand({ FunctionName: name }),
        );
        return Configuration;
    }
    throw caught;
}
};

// Function creation fails if the Role is not ready. This retries
// function creation until it succeeds or it times out.
const { FunctionArn } = await retry(
    { intervalInMs: 1000, maxRetries: 60 },
    createFunction,
);

return {
    arn: FunctionArn,
    cleanUp: async () => {
        await lambdaClient.send(
            new DeleteFunctionCommand({ FunctionName: name }),
        );
    },
};
}

/**
 * This uploads some sample coordinate data to an Amazon S3 bucket.
 * The Amazon SageMaker Geospatial vector enrichment job will take the simple Lat/
Long
 * coordinates in this file and augment them with more detailed location data.
 * @param {{bucketName: string, s3Client: import('@aws-sdk/client-s3').S3Client}}
props
 */
export async function uploadCSVDataToS3({ bucketName, s3Client }) {
    const s3Path = `${dirnameFromMetaUrl(
        import.meta.url,
    )}../../../../../scenarios/features/sagemaker_pipelines/resources/
latlongtest.csv`;

    await s3Client.send(
        new PutObjectCommand({

```

```

        Bucket: bucketName,
        Key: "input/sample_data.csv",
        Body: readFileSync(s3Path),
    }},
    );
}

/**
 * Create the AWS IAM role that will be assumed by the Amazon SageMaker pipeline.
 * @param {{name: string, iamClient: import('@aws-sdk/client-iam').IAMClient, wait:
 * (ms: number) => Promise<void>}} props
 */
export async function createSagemakerRole({ name, iamClient, wait }) {
    let role = null;

    const createRole = () =>
        iamClient.send(
            new CreateRoleCommand({
                RoleName: name,
                AssumeRolePolicyDocument: JSON.stringify({
                    Version: "2012-10-17",
                    Statement: [
                        {
                            Effect: "Allow",
                            Action: ["sts:AssumeRole"],
                            Principal: {
                                Service: [
                                    "sagemaker.amazonaws.com",
                                    "sagemaker-geospatial.amazonaws.com",
                                ],
                            },
                        },
                    ],
                }),
            }),
        );

    try {
        const { Role } = await createRole();
        role = Role;
        // Wait for the role to be ready.
        await wait(10);
    } catch (caught) {
        if (

```

```

        caught instanceof Error &&
        caught.name === "EntityAlreadyExistsException"
    ) {
        const { Role } = await iamClient.send(
            new GetRoleCommand({ RoleName: name }),
        );
        role = Role;
    } else {
        throw caught;
    }
}

return {
    arn: role.Arn,
    cleanUp: async () => {
        await iamClient.send(new DeleteRoleCommand({ RoleName: name }));
    },
};
}

/**
 * Create the Amazon SageMaker execution policy. This policy grants permission to
 * invoke the AWS Lambda function, read/write to the Amazon S3 bucket, and send
 * messages to
 * the Amazon SQS queue.
 * @param {{ name: string, sqsQueueArn: string, lambdaArn: string, iamClient:
 * import('@aws-sdk/client-iam').IAMClient, s3BucketName: string}} props
 */
export async function createSagemakerExecutionPolicy({
    sqsQueueArn,
    lambdaArn,
    iamClient,
    name,
    s3BucketName,
}) {
    const policyConfig = {
        Version: "2012-10-17",
        Statement: [
            {
                Effect: "Allow",
                Action: ["lambda:InvokeFunction"],
                Resource: lambdaArn,
            },
            {

```



```

    Effect: "Allow",
    Action: ["s3:*"],
    Resource: [
      `arn:aws:s3:::${s3BucketName}`,
      `arn:aws:s3:::${s3BucketName}/*`,
    ],
  },
  {
    Effect: "Allow",
    Action: ["sqs:SendMessage"],
    Resource: sqsQueueArn,
  },
],
};

const createPolicy = () =>
  iamClient.send(
    new CreatePolicyCommand({
      PolicyDocument: JSON.stringify(policyConfig),
      PolicyName: name,
    }),
  );

let policy = null;

try {
  const { Policy } = await createPolicy();
  policy = Policy;
} catch (caught) {
  if (
    caught instanceof Error &&
    caught.name === "EntityAlreadyExistsException"
  ) {
    const { Policies } = await iamClient.send(new ListPoliciesCommand({}));
    if (Policies) {
      policy = Policies.find((p) => p.PolicyName === name);
    } else {
      throw new Error("No policies found.");
    }
  } else {
    throw caught;
  }
}

```

```

    return {
      arn: policy?.Arn,
      policyConfig,
      cleanUp: async () => {
        await iamClient.send(new DeletePolicyCommand({ PolicyArn: policy?.Arn }));
      },
    };
  }

  /**
   * Create the Amazon SageMaker pipeline using a JSON pipeline definition. The
   * definition
   * can also be provided as an Amazon S3 object using PipelineDefinitionS3Location.
   * @param {{roleArn: string, name: string, sagemakerClient: import('@aws-sdk/client-
   * sagemaker').SageMakerClient}} props
   */
  export async function createSagemakerPipeline({
    // Assumes an AWS IAM role has been created for this pipeline.
    roleArn,
    name,
    // Assumes an AWS Lambda function has been created for this pipeline.
    functionArn,
    sagemakerClient,
  }) {
    const pipelineDefinition = readFileSync(
      // dirnameFromMetaUrl is a local utility function. You can find its
      implementation
      // on GitHub.
      `${dirnameFromMetaUrl(
        import.meta.url,
      )}../../../../scenarios/features/sagemaker_pipelines/resources/
      GeoSpatialPipeline.json`,
    )
      .toString()
      .replace(/.*FUNCTION_ARN.*/g, functionArn);

    let arn = null;

    const createPipeline = () =>
      sagemakerClient.send(
        new CreatePipelineCommand({
          PipelineName: name,
          PipelineDefinition: pipelineDefinition,
          RoleArn: roleArn,

```

```

    }),
  );

  try {
    const { PipelineArn } = await createPipeline();
    arn = PipelineArn;
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "ValidationException" &&
      caught.message.includes(
        "Pipeline names must be unique within an AWS account and region",
      )
    ) {
      const { PipelineArn } = await sagemakerClient.send(
        new DescribePipelineCommand({ PipelineName: name }),
      );
      arn = PipelineArn;
    } else {
      throw caught;
    }
  }

  return {
    arn,
    cleanUp: async () => {
      await sagemakerClient.send(
        new DeletePipelineCommand({ PipelineName: name }),
      );
    },
  };
}

/**
 * Create an Amazon SQS queue. The Amazon SageMaker pipeline will send messages
 * to this queue that are then processed by the AWS Lambda function.
 * @param {{name: string, sqsClient: import('@aws-sdk/client-sqs').SQSClient}} props
 */
export async function createSQSQueue({ name, sqsClient }) {
  const createSqsQueue = () =>
    sqsClient.send(
      new CreateQueueCommand({
        QueueName: name,
        Attributes: {

```

```

        DelaySeconds: "5",
        ReceiveMessageWaitTimeSeconds: "5",
        VisibilityTimeout: "300",
    },
 )),
);

let queueUrl = null;
try {
  const { QueueUrl } = await createSqsQueue();
  queueUrl = QueueUrl;
} catch (caught) {
  if (caught instanceof Error && caught.name === "QueueNameExists") {
    const { QueueUrl } = await sqsClient.send(
      new GetQueueUrlCommand({ QueueName: name }),
    );
    queueUrl = QueueUrl;
  } else {
    throw caught;
  }
}

const { Attributes } = await retry(
  { intervalInMs: 1000, maxRetries: 60 },
  () =>
    sqsClient.send(
      new GetQueueAttributesCommand({
        QueueUrl: queueUrl,
        AttributeNames: ["QueueArn"],
      }),
    ),
);

return {
  queueUrl,
  queueArn: Attributes.QueueArn,
  cleanUp: async () => {
    await sqsClient.send(new DeleteQueueCommand({ QueueUrl: queueUrl }));
  },
};
}

/**
 * Configure the AWS Lambda function to long poll for messages from the Amazon SQS

```

```

* queue.
* @param {{
*   paginateListEventSourceMappings: () => Generator<import('@aws-sdk/client-
lambda').ListEventSourceMappingsCommandOutput>,
*   lambdaName: string,
*   queueArn: string,
*   lambdaClient: import('@aws-sdk/client-lambda').LambdaClient}} props
*/
export async function configureLambdaSQSEventSource({
  lambdaName,
  queueArn,
  lambdaClient,
  paginateListEventSourceMappings,
}) {
  let uuid = null;
  const createEvenSourceMapping = () =>
    lambdaClient.send(
      new CreateEventSourceMappingCommand({
        EventSourceArn: queueArn,
        FunctionName: lambdaName,
      }),
    );

  try {
    const { UUID } = await createEvenSourceMapping();
    uuid = UUID;
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "ResourceConflictException"
    ) {
      const paginator = paginateListEventSourceMappings(
        { client: lambdaClient },
        {},
      );
      /**
       * @type {import('@aws-sdk/client-lambda').EventSourceMappingConfiguration[]}
       */
      const eventSourceMappings = [];
      for await (const page of paginator) {
        eventSourceMappings.concat(page.EventSourceMappings || []);
      }

      const { Configuration } = await lambdaClient.send(

```

```

        new GetFunctionCommand({ FunctionName: lambdaName })),
    );

    uuid = eventSourceMappings.find(
      (mapping) =>
        mapping.EventSourceArn === queueArn &&
        mapping.FunctionArn === Configuration.FunctionArn,
    ).UUID;
  } else {
    throw caught;
  }
}

return {
  cleanUp: async () => {
    await lambdaClient.send(
      new DeleteEventSourceMappingCommand({
        UUID: uuid,
      }),
    );
  },
};
}

/**
 * Create an Amazon S3 bucket that will store the simple coordinate file as input
 * and the output of the Amazon SageMaker Geospatial vector enrichment job.
 * @param {{
 *   s3Client: import('@aws-sdk/client-s3').S3Client,
 *   name: string,
 *   paginateListObjectsV2: () => Generator<import('@aws-sdk/client-s3').ListObjectsCommandOutput>
 * }} props
 */
export async function createS3Bucket({
  name,
  s3Client,
  paginateListObjectsV2,
}) {
  await s3Client.send(new CreateBucketCommand({ Bucket: name }));

  return {
    cleanUp: async () => {
      const paginator = paginateListObjectsV2(

```

```

    { client: s3Client },
    { Bucket: name },
  );
  for await (const page of paginator) {
    const objects = page.Contents;
    if (objects) {
      for (const object of objects) {
        await s3Client.send(
          new DeleteObjectCommand({ Bucket: name, Key: object.Key }),
        );
      }
    }
  }
  await s3Client.send(new DeleteBucketCommand({ Bucket: name }));
},
};
}

/**
 * Start the execution of the Amazon SageMaker pipeline. Parameters that are
 * passed in are used in the AWS Lambda function.
 * @param {{
 *   name: string,
 *   sagemakerClient: import('@aws-sdk/client-sagemaker').SageMakerClient,
 *   roleArn: string,
 *   queueUrl: string,
 *   s3InputBucketName: string,
 * }} props
 */
export async function startPipelineExecution({
  sagemakerClient,
  name,
  bucketName,
  roleArn,
  queueUrl,
}) {
  /**
   * The Vector Enrichment Job requests CSV data. This configuration points to a CSV
   * file in an Amazon S3 bucket.
   * @type {import("@aws-sdk/client-sagemaker-geospatial").VectorEnrichmentJobInputConfig}
   */
  const inputConfig = {
    DataSourceConfig: {

```

```

    S3Data: {
      S3Uri: `s3://${bucketName}/input/sample_data.csv`,
    },
  },
  DocumentType: VectorEnrichmentJobDocumentType.CSV,
};

/**
 * The Vector Enrichment Job adds additional data to the source CSV. This
configuration points
 * to an Amazon S3 prefix where the output will be stored.
 * @type {import("@aws-sdk/client-sagemaker-geospatial").ExportVectorEnrichmentJobOutputConfig}
 */
const outputConfig = {
  S3Data: {
    S3Uri: `s3://${bucketName}/output/`,
  },
};

/**
 * This job will be a Reverse Geocoding Vector Enrichment Job. Reverse Geocoding
requires
 * latitude and longitude values.
 * @type {import("@aws-sdk/client-sagemaker-geospatial").VectorEnrichmentJobConfig}
 */
const jobConfig = {
  ReverseGeocodingConfig: {
    XAttributeName: "Longitude",
    YAttributeName: "Latitude",
  },
};

const { PipelineExecutionArn } = await sagemakerClient.send(
  new StartPipelineExecutionCommand({
    PipelineName: name,
    PipelineExecutionDisplayName: `${name}-example-execution`,
    PipelineParameters: [
      { Name: "parameter_execution_role", Value: roleArn },
      { Name: "parameter_queue_url", Value: queueUrl },
      {
        Name: "parameter_vej_input_config",
        Value: JSON.stringify(inputConfig),
      },
    ],
  })
);

```



```

    },
    {
      Name: "parameter_vej_export_config",
      Value: JSON.stringify(outputConfig),
    },
    {
      Name: "parameter_step_1_vej_config",
      Value: JSON.stringify(jobConfig),
    },
  ],
  )),
);

return {
  arn: PipelineExecutionArn,
};
}

/**
 * Poll the executing pipeline until the status is 'SUCCEEDED', 'STOPPED', or
 * 'FAILED'.
 * @param {{ arn: string, sagemakerClient: import('@aws-sdk/client-sagemaker').SageMakerClient, wait: (ms: number) => Promise<void> }} props
 */
export async function waitForPipelineComplete({ arn, sagemakerClient, wait }) {
  const command = new DescribePipelineExecutionCommand({
    PipelineExecutionArn: arn,
  });

  let complete = false;
  const intervalInSeconds = 15;
  const COMPLETION_STATUSES = [
    PipelineExecutionStatus.FAILED,
    PipelineExecutionStatus.STOPPED,
    PipelineExecutionStatus.SUCCEEDED,
  ];

  do {
    const { PipelineExecutionStatus: status, FailureReason } =
      await sagemakerClient.send(command);

    complete = COMPLETION_STATUSES.includes(status);

    if (!complete) {

```

```

        console.log(
            `Pipeline is ${status}. Waiting ${intervalInSeconds} seconds before checking
            again.` ,
        );
        await wait(intervalInSeconds);
    } else if (status === PipelineExecutionStatus.FAILED) {
        throw new Error(`Pipeline failed because: ${FailureReason}`);
    } else if (status === PipelineExecutionStatus.STOPPED) {
        throw new Error("Pipeline was forcefully stopped.");
    } else {
        console.log(`Pipeline execution ${status}.`);
    }
} while (!complete);
}

/**
 * Return the string value of an Amazon S3 object.
 * @param {{ bucket: string, key: string, s3Client: import('@aws-sdk/client-s3').S3Client }} param0
 */
export async function getObject({ bucket, s3Client }) {
    const prefix = "output/";
    const { Contents } = await s3Client.send(
        new ListObjectsV2Command({ MaxKeys: 1, Bucket: bucket, Prefix: prefix }),
    );

    if (!Contents.length) {
        throw new Error("No objects found in bucket.");
    }

    // Find the CSV file.
    const outputObject = Contents.find((obj) => obj.Key.endsWith(".csv"));

    if (!outputObject) {
        throw new Error(`No CSV file found in bucket with the prefix "${prefix}".`);
    }

    const { Body } = await s3Client.send(
        new GetObjectCommand({
            Bucket: bucket,
            Key: outputObject.Key,
        }),
    );
};

```

```
    return Body.toString();
  }
```

Diese Funktion ist ein Auszug aus einer Datei, die die vorherigen Bibliotheksfunktionen verwendet, um eine SageMaker AI-Pipeline einzurichten, auszuführen und alle erstellten Ressourcen zu löschen.

```
import { retry, wait } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";
import {
  attachPolicy,
  configureLambdaSQSEventSource,
  createLambdaExecutionPolicy,
  createLambdaExecutionRole,
  createLambdaFunction,
  createLambdaLayer,
  createS3Bucket,
  createSQSQueue,
  createSagemakerExecutionPolicy,
  createSagemakerPipeline,
  createSagemakerRole,
  getObject,
  startPipelineExecution,
  uploadCSVDataToS3,
  waitForPipelineComplete,
} from "./lib.js";
import { MESSAGES } from "./messages.js";

export class SageMakerPipelinesWkflw {
  names = {
    LAMBDA_EXECUTION_ROLE: "sagemaker-wkflw-lambda-execution-role",
    LAMBDA_EXECUTION_ROLE_POLICY:
      "sagemaker-wkflw-lambda-execution-role-policy",
    LAMBDA_FUNCTION: "sagemaker-wkflw-lambda-function",
    LAMBDA_LAYER: "sagemaker-wkflw-lambda-layer",
    SAGE_MAKER_EXECUTION_ROLE: "sagemaker-wkflw-pipeline-execution-role",
    SAGE_MAKER_EXECUTION_ROLE_POLICY:
      "sagemaker-wkflw-pipeline-execution-role-policy",
    SAGE_MAKER_PIPELINE: "sagemaker-wkflw-pipeline",
    SQS_QUEUE: "sagemaker-wkflw-sqs-queue",
    S3_BUCKET: `sagemaker-wkflw-s3-bucket-${Date.now()}`,
  };
}
```

```
cleanUpFunctions = [];

/**
 * @param {import("@aws-doc-sdk-examples/lib/prompter.js").Prompter} prompter
 * @param {import("@aws-doc-sdk-examples/lib/logger.js").Logger} logger
 * @param {{ IAM: import("@aws-sdk/client-iam").IAMClient, Lambda: import("@aws-
sdk/client-lambda").LambdaClient, SageMaker: import("@aws-sdk/client-
sagemaker").SageMakerClient, S3: import("@aws-sdk/client-s3").S3Client, SQS:
import("@aws-sdk/client-sqs").SQSClient }} clients
 */
constructor(prompter, logger, clients) {
  this.prompter = prompter;
  this.logger = logger;
  this.clients = clients;
}

async run() {
  try {
    await this.startWorkflow();
  } catch (err) {
    console.error(err);
    throw err;
  } finally {
    this.logger.logSeparator();
    const doCleanUp = await this.prompter.confirm({
      message: "Clean up resources?",
    });
    if (doCleanUp) {
      await this.cleanUp();
    }
  }
}

async cleanUp() {
  // Run all of the clean up functions. If any fail, we log the error and
  continue.
  // This ensures all clean up functions are run.
  for (let i = this.cleanUpFunctions.length - 1; i >= 0; i--) {
    await retry(
      { intervalInMs: 1000, maxRetries: 60, swallowError: true },
      this.cleanUpFunctions[i],
    );
  }
}
```

```
async startWorkflow() {
  this.logger.logSeparator(MESSAGES.greetingHeader);
  await this.logger.log(MESSAGES.greeting);

  this.logger.logSeparator();
  await this.logger.log(
    MESSAGES.creatingRole.replace(
      "${ROLE_NAME}",
      this.names.LAMBDA_EXECUTION_ROLE,
    ),
  );

  // Create an IAM role that will be assumed by the AWS Lambda function. This
function
  // is triggered by Amazon SQS messages and calls SageMaker and SageMaker
GeoSpatial actions.
  const { arn: lambdaExecutionRoleArn, cleanUp: lambdaExecutionRoleCleanUp } =
    await createLambdaExecutionRole({
      name: this.names.LAMBDA_EXECUTION_ROLE,
      iamClient: this.clients.IAM,
    });
  // Add a clean up step to a stack for every resource created.
  this.cleanUpFunctions.push(lambdaExecutionRoleCleanUp);

  await this.logger.log(
    MESSAGES.roleCreated.replace(
      "${ROLE_NAME}",
      this.names.LAMBDA_EXECUTION_ROLE,
    ),
  );

  this.logger.logSeparator();

  await this.logger.log(
    MESSAGES.creatingRole.replace(
      "${ROLE_NAME}",
      this.names.SAGE_MAKER_EXECUTION_ROLE,
    ),
  );

  // Create an IAM role that will be assumed by the SageMaker pipeline. The
pipeline
  // sends messages to an Amazon SQS queue and puts/retrieves Amazon S3 objects.
```

```
const {
  arn: pipelineExecutionRoleArn,
  cleanUp: pipelineExecutionRoleCleanUp,
} = await createSagemakerRole({
  iamClient: this.clients.IAM,
  name: this.names.SAGE_MAKER_EXECUTION_ROLE,
  wait,
});
this.cleanUpFunctions.push(pipelineExecutionRoleCleanUp);

await this.logger.log(
  MESSAGES.roleCreated.replace(
    "${ROLE_NAME}",
    this.names.SAGE_MAKER_EXECUTION_ROLE,
  ),
);

this.logger.logSeparator();

// Create an IAM policy that allows the AWS Lambda function to invoke SageMaker APIs.
const {
  arn: lambdaExecutionPolicyArn,
  policy: lambdaPolicy,
  cleanUp: lambdaExecutionPolicyCleanUp,
} = await createLambdaExecutionPolicy({
  name: this.names.LAMBDA_EXECUTION_ROLE_POLICY,
  s3BucketName: this.names.S3_BUCKET,
  iamClient: this.clients.IAM,
  pipelineExecutionRoleArn,
});
this.cleanUpFunctions.push(lambdaExecutionPolicyCleanUp);

console.log(JSON.stringify(lambdaPolicy, null, 2), "\n");

await this.logger.log(
  MESSAGES.attachPolicy
    .replace("${POLICY_NAME}", this.names.LAMBDA_EXECUTION_ROLE_POLICY)
    .replace("${ROLE_NAME}", this.names.LAMBDA_EXECUTION_ROLE),
);

await this.prompter.checkContinue();

// Attach the Lambda execution policy to the execution role.
```

```
const { cleanUp: lambdaExecutionRolePolicyCleanUp } = await attachPolicy({
  roleName: this.names.LAMBDA_EXECUTION_ROLE,
  policyArn: lambdaExecutionPolicyArn,
  iamClient: this.clients.IAM,
});
this.cleanUpFunctions.push(lambdaExecutionRolePolicyCleanUp);

await this.logger.log(MESSAGES.policyAttached);

this.logger.logSeparator();

// Create Lambda layer for SageMaker packages.
const { versionArn: layerVersionArn, cleanUp: lambdaLayerCleanUp } =
  await createLambdaLayer({
    name: this.names.LAMBDA_LAYER,
    lambdaClient: this.clients.Lambda,
  });
this.cleanUpFunctions.push(lambdaLayerCleanUp);

await this.logger.log(
  MESSAGES.creatingFunction.replace(
    "${FUNCTION_NAME}",
    this.names.LAMBDA_FUNCTION,
  ),
);

// Create the Lambda function with the execution role.
const { arn: lambdaArn, cleanUp: lambdaCleanUp } =
  await createLambdaFunction({
    roleArn: lambdaExecutionRoleArn,
    lambdaClient: this.clients.Lambda,
    name: this.names.LAMBDA_FUNCTION,
    layerVersionArn,
  });
this.cleanUpFunctions.push(lambdaCleanUp);

await this.logger.log(
  MESSAGES.functionCreated.replace(
    "${FUNCTION_NAME}",
    this.names.LAMBDA_FUNCTION,
  ),
);

this.logger.logSeparator();
```

```
await this.logger.log(
  MESSAGES.creatingSQSQueue.replace("${QUEUE_NAME}", this.names.SQS_QUEUE),
);

// Create an SQS queue for the SageMaker pipeline.
const {
  queueUrl,
  queueArn,
  cleanUp: queueCleanUp,
} = await createSQSQueue({
  name: this.names.SQS_QUEUE,
  sqsClient: this.clients.SQS,
});
this.cleanUpFunctions.push(queueCleanUp);

await this.logger.log(
  MESSAGES.sqsQueueCreated.replace("${QUEUE_NAME}", this.names.SQS_QUEUE),
);

this.logger.logSeparator();

await this.logger.log(
  MESSAGES.configuringLambdaSQSEventSource
    .replace("${LAMBDA_NAME}", this.names.LAMBDA_FUNCTION)
    .replace("${QUEUE_NAME}", this.names.SQS_QUEUE),
);

// Configure the SQS queue as an event source for the Lambda.
const { cleanUp: lambdaSQSEventSourceCleanUp } =
  await configureLambdaSQSEventSource({
    lambdaArn,
    lambdaName: this.names.LAMBDA_FUNCTION,
    queueArn,
    sqsClient: this.clients.SQS,
    lambdaClient: this.clients.Lambda,
  });
this.cleanUpFunctions.push(lambdaSQSEventSourceCleanUp);

await this.logger.log(
  MESSAGES.lambdaSQSEventSourceConfigured
    .replace("${LAMBDA_NAME}", this.names.LAMBDA_FUNCTION)
    .replace("${QUEUE_NAME}", this.names.SQS_QUEUE),
);
```



```
this.logger.logSeparator();

// Create an IAM policy that allows the SageMaker pipeline to invoke AWS Lambda
// and send messages to the Amazon SQS queue.
const {
  arn: pipelineExecutionPolicyArn,
  policy: sagemakerPolicy,
  cleanUp: pipelineExecutionPolicyCleanUp,
} = await createSagemakerExecutionPolicy({
  sqsQueueArn: queueArn,
  lambdaArn,
  iamClient: this.clients.IAM,
  name: this.names.SAGE_MAKER_EXECUTION_ROLE_POLICY,
  s3BucketName: this.names.S3_BUCKET,
});
this.cleanUpFunctions.push(pipelineExecutionPolicyCleanUp);

console.log(JSON.stringify(sagemakerPolicy, null, 2));

await this.logger.log(
  MESSAGES.attachPolicy
    .replace("${POLICY_NAME}", this.names.SAGE_MAKER_EXECUTION_ROLE_POLICY)
    .replace("${ROLE_NAME}", this.names.SAGE_MAKER_EXECUTION_ROLE),
);

await this.prompter.checkContinue();

// Attach the SageMaker execution policy to the execution role.
const { cleanUp: pipelineExecutionRolePolicyCleanUp } = await attachPolicy({
  roleName: this.names.SAGE_MAKER_EXECUTION_ROLE,
  policyArn: pipelineExecutionPolicyArn,
  iamClient: this.clients.IAM,
});
this.cleanUpFunctions.push(pipelineExecutionRolePolicyCleanUp);
// Wait for the role to be ready. If the role is used immediately,
// the pipeline will fail.
await wait(5);

await this.logger.log(MESSAGES.policyAttached);

this.logger.logSeparator();

await this.logger.log(
```

```
    MESSAGES.creatingPipeline.replace(
      "${PIPELINE_NAME}",
      this.names.SAGE_MAKER_PIPELINE,
    ),
  );
};

// Create the SageMaker pipeline.
const { cleanUp: pipelineCleanUp } = await createSagemakerPipeline({
  roleArn: pipelineExecutionRoleArn,
  functionArn: lambdaArn,
  sagemakerClient: this.clients.SageMaker,
  name: this.names.SAGE_MAKER_PIPELINE,
});
this.cleanUpFunctions.push(pipelineCleanUp);

await this.logger.log(
  MESSAGES.pipelineCreated.replace(
    "${PIPELINE_NAME}",
    this.names.SAGE_MAKER_PIPELINE,
  ),
);

this.logger.logSeparator();

await this.logger.log(
  MESSAGES.creatingS3Bucket.replace("${BUCKET_NAME}", this.names.S3_BUCKET),
);

// Create an S3 bucket for storing inputs and outputs.
const { cleanUp: s3BucketCleanUp } = await createS3Bucket({
  name: this.names.S3_BUCKET,
  s3Client: this.clients.S3,
});
this.cleanUpFunctions.push(s3BucketCleanUp);

await this.logger.log(
  MESSAGES.s3BucketCreated.replace("${BUCKET_NAME}", this.names.S3_BUCKET),
);

this.logger.logSeparator();

await this.logger.log(
  MESSAGES.uploadingInputData.replace(
    "${BUCKET_NAME}",
```

```
        this.names.S3_BUCKET,
    ),
);

// Upload CSV Lat/Long data to S3.
await uploadCSVDataToS3({
    bucketName: this.names.S3_BUCKET,
    s3Client: this.clients.S3,
});

await this.logger.log(MESSAGES.inputDataUploaded);

this.logger.logSeparator();

await this.prompter.checkContinue(MESSAGES.executePipeline);

// Execute the SageMaker pipeline.
const { arn: pipelineExecutionArn } = await startPipelineExecution({
    name: this.names.SAGE_MAKER_PIPELINE,
    sagemakerClient: this.clients.SageMaker,
    roleArn: pipelineExecutionRoleArn,
    bucketName: this.names.S3_BUCKET,
    queueUrl,
});

// Wait for the pipeline execution to finish.
await waitForPipelineComplete({
    arn: pipelineExecutionArn,
    sagemakerClient: this.clients.SageMaker,
    wait,
});

this.logger.logSeparator();

await this.logger.log(MESSAGES.outputDelay);

// The getOutput function will throw an error if the output is not
// found. The retry function will retry a failed function call once
// ever 10 seconds for 2 minutes.
const output = await retry({ intervalInMs: 10000, maxRetries: 12 }, () =>
    getObject({
        bucket: this.names.S3_BUCKET,
        s3Client: this.clients.S3,
    })),
```

```
);  
  
this.logger.logSeparator();  
await this.logger.log(MESSAGES.outputDataRetrieved);  
console.log(output.split("\n").slice(0, 6).join("\n"));  
}  
}
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für JavaScript - API-Referenz.
 - [CreatePipeline](#)
 - [DeletePipeline](#)
 - [DescribePipelineExecution](#)
 - [StartPipelineExecution](#)
 - [UpdatePipeline](#)

Secrets Manager Manager-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie AWS SDK für JavaScript (v3) mit Secrets Manager verwenden.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kodex finden.

Themen

- [Aktionen](#)

Aktionen

GetSecretValue

Das folgende Codebeispiel zeigt die Verwendung von `GetSecretValue`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import {
  GetSecretValueCommand,
  SecretsManagerClient,
} from "@aws-sdk/client-secrets-manager";

export const getSecretValue = async (secretName = "SECRET_NAME") => {
  const client = new SecretsManagerClient();
  const response = await client.send(
    new GetSecretValueCommand({
      SecretId: secretName,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '584eb612-f8b0-48c9-855e-6d246461b604',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   ARN: 'arn:aws:secretsmanager:us-east-1:xxxxxxxxxxxx:secret:binary-
secret-3873048-xxxxxx',
  //   CreatedDate: 2023-08-08T19:29:51.294Z,
  //   Name: 'binary-secret-3873048',
  //   SecretBinary: Uint8Array(11) [
  //     98, 105, 110, 97, 114,
  //     121, 32, 100, 97, 116,
  //     97
  //   ],
  //   VersionId: '712083f4-0d26-415e-8044-16735142cd6a',
  //   VersionStages: [ 'AWSCURRENT' ]
  // }
```

```
if (response.SecretString) {  
    return response.SecretString;  
}  
  
if (response.SecretBinary) {  
    return response.SecretBinary;  
}  
};
```

- Einzelheiten zur API finden Sie [GetSecretValue](#) in der AWS SDK für JavaScript API-Referenz.

Amazon SES SES-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK für JavaScript (v3) mit Amazon SES Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien anzeigen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb eines Service aufrufen oder mit anderen AWS-Services kombinieren.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kodex finden.

Themen

- [Aktionen](#)
- [Szenarien](#)

Aktionen

CreateReceiptFilter

Das folgende Codebeispiel zeigt, wie Sie `CreateReceiptFilter`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import {
  CreateReceiptFilterCommand,
  ReceiptFilterPolicy,
} from "@aws-sdk/client-ses";
import { sesClient } from "../libs/sesClient.js";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";

const createCreateReceiptFilterCommand = ({ policy, ipOrRange, name }) => {
  return new CreateReceiptFilterCommand({
    Filter: {
      IpFilter: {
        Cidr: ipOrRange, // string, either a single IP address (10.0.0.1) or an IP
                        // address range in CIDR notation (10.0.0.1/24)).
        Policy: policy, // enum ReceiptFilterPolicy, email traffic from the filtered
                        // addressesOptions.
      },
      /*
       * The name of the IP address filter. Only ASCII letters, numbers, underscores,
       * or dashes.
       * Must be less than 64 characters and start and end with a letter or number.
       */
      Name: name,
    },
  });
};

const FILTER_NAME = getUniqueName("ReceiptFilter");

const run = async () => {
  const createReceiptFilterCommand = createCreateReceiptFilterCommand({
    policy: ReceiptFilterPolicy.Allow,
    ipOrRange: "10.0.0.1",
    name: FILTER_NAME,
  });
};
```

```

try {
  return await sesClient.send(createReceiptFilterCommand);
} catch (caught) {
  if (caught instanceof Error && caught.name === "MessageRejected") {
    /** @type { import('@aws-sdk/client-ses').MessageRejected } */
    const messageRejectedError = caught;
    return messageRejectedError;
  }
  throw caught;
}
};

```

- Einzelheiten zur API finden Sie [CreateReceiptFilter](#) in der AWS SDK für JavaScript API-Referenz.

CreateReceiptRule

Das folgende Codebeispiel zeigt die Verwendung `CreateReceiptRule`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```

import { CreateReceiptRuleCommand, TlsPolicy } from "@aws-sdk/client-ses";
import { sesClient } from "../libs/sesClient.js";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";

const RULE_SET_NAME = getUniqueName("RuleSetName");
const RULE_NAME = getUniqueName("RuleName");
const S3_BUCKET_NAME = getUniqueName("S3BucketName");

const createS3ReceiptRuleCommand = ({
  bucketName,
  emailAddresses,

```



```

    name,
    ruleSet,
  }) => {
    return new CreateReceiptRuleCommand({
      Rule: {
        Actions: [
          {
            S3Action: {
              BucketName: bucketName,
              ObjectKeyPrefix: "email",
            },
          },
        ],
        Recipients: emailAddresses,
        Enabled: true,
        Name: name,
        ScanEnabled: false,
        TlsPolicy: TlsPolicy.Optional,
      },
      RuleSetName: ruleSet, // Required
    });
  };

const run = async () => {
  const s3ReceiptRuleCommand = createS3ReceiptRuleCommand({
    bucketName: S3_BUCKET_NAME,
    emailAddresses: ["email@example.com"],
    name: RULE_NAME,
    ruleSet: RULE_SET_NAME,
  });

  try {
    return await sesClient.send(s3ReceiptRuleCommand);
  } catch (err) {
    console.log("Failed to create S3 receipt rule.", err);
    throw err;
  }
};

```

- Einzelheiten zur API finden Sie [CreateReceiptRule](#) in der AWS SDK für JavaScript API-Referenz.

CreateReceiptRuleSet

Das folgende Codebeispiel zeigt die Verwendung `CreateReceiptRuleSet`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { CreateReceiptRuleSetCommand } from "@aws-sdk/client-ses";
import { sesClient } from "../libs/sesClient.js";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";

const RULE_SET_NAME = getUniqueName("RuleSetName");

const createCreateReceiptRuleSetCommand = (ruleSetName) => {
  return new CreateReceiptRuleSetCommand({ RuleSetName: ruleSetName });
};

const run = async () => {
  const createReceiptRuleSetCommand =
    createCreateReceiptRuleSetCommand(RULE_SET_NAME);

  try {
    return await sesClient.send(createReceiptRuleSetCommand);
  } catch (err) {
    console.log("Failed to create receipt rule set", err);
    return err;
  }
};
```

- Einzelheiten zur API finden Sie [CreateReceiptRuleSet](#) in der AWS SDK für JavaScript API-Referenz.

CreateTemplate

Das folgende Codebeispiel zeigt die Verwendung `CreateTemplate`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { CreateTemplateCommand } from "@aws-sdk/client-ses";
import { sesClient } from "../libs/sesClient.js";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";

const TEMPLATE_NAME = getUniqueName("TestTemplateName");

const createCreateTemplateCommand = () => {
  return new CreateTemplateCommand({
    /**
     * The template feature in Amazon SES is based on the Handlebars template
     * system.
     */
    Template: {
      /**
       * The name of an existing template in Amazon SES.
       */
      TemplateName: TEMPLATE_NAME,
      HtmlPart: `
        <h1>Hello, {{contact.firstName}}!</h1>
        <p>
          Did you know Amazon has a mascot named Peccy?
        </p>
      `,
      SubjectPart: "Amazon Tip",
    },
  });
};

const run = async () => {
  const createTemplateCommand = createCreateTemplateCommand();

  try {
    return await sesClient.send(createTemplateCommand);
  }
}
```

```
    } catch (err) {  
      console.log("Failed to create template.", err);  
      return err;  
    }  
  };  
};
```

- Einzelheiten zur API finden Sie [CreateTemplate](#) in der AWS SDK für JavaScript API-Referenz.

DeleteIdentity

Das folgende Codebeispiel zeigt die Verwendung `DeleteIdentity`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { DeleteIdentityCommand } from "@aws-sdk/client-ses";  
import { sesClient } from "../libs/sesClient.js";  
  
const IDENTITY_EMAIL = "fake@example.com";  
  
const createDeleteIdentityCommand = (identityName) => {  
  return new DeleteIdentityCommand({  
    Identity: identityName,  
  });  
};  
  
const run = async () => {  
  const deleteIdentityCommand = createDeleteIdentityCommand(IDENTITY_EMAIL);  
  
  try {  
    return await sesClient.send(deleteIdentityCommand);  
  } catch (err) {  
    console.log("Failed to delete identity.", err);  
    return err;  
  }  
};
```

- Einzelheiten zur API finden Sie [DeleteIdentity](#) in der AWS SDK für JavaScript API-Referenz.

DeleteReceiptFilter

Das folgende Codebeispiel zeigt die Verwendung `DeleteReceiptFilter`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { DeleteReceiptFilterCommand } from "@aws-sdk/client-ses";
import { sesClient } from "../libs/sesClient.js";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utis/util-string.js";

const RECEIPT_FILTER_NAME = getUniqueName("ReceiptFilterName");

const createDeleteReceiptFilterCommand = (filterName) => {
  return new DeleteReceiptFilterCommand({ FilterName: filterName });
};

const run = async () => {
  const deleteReceiptFilterCommand =
    createDeleteReceiptFilterCommand(RECEIPT_FILTER_NAME);

  try {
    return await sesClient.send(deleteReceiptFilterCommand);
  } catch (err) {
    console.log("Error deleting receipt filter.", err);
    return err;
  }
};
```

- Einzelheiten zur API finden Sie [DeleteReceiptFilter](#) in der AWS SDK für JavaScript API-Referenz.

DeleteReceiptRule

Das folgende Codebeispiel zeigt die Verwendung `DeleteReceiptRule`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { DeleteReceiptRuleCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "../libs/sesClient.js";

const RULE_NAME = getUniqueName("RuleName");
const RULE_SET_NAME = getUniqueName("RuleSetName");

const createDeleteReceiptRuleCommand = () => {
  return new DeleteReceiptRuleCommand({
    RuleName: RULE_NAME,
    RuleSetName: RULE_SET_NAME,
  });
};

const run = async () => {
  const deleteReceiptRuleCommand = createDeleteReceiptRuleCommand();
  try {
    return await sesClient.send(deleteReceiptRuleCommand);
  } catch (err) {
    console.log("Failed to delete receipt rule.", err);
    return err;
  }
};
```

- Einzelheiten zur API finden Sie [DeleteReceiptRule](#) in der AWS SDK für JavaScript API-Referenz.

DeleteReceiptRuleSet

Das folgende Codebeispiel zeigt die Verwendung `DeleteReceiptRuleSet`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { DeleteReceiptRuleSetCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "../libs/sesClient.js";

const RULE_SET_NAME = getUniqueName("RuleSetName");

const createDeleteReceiptRuleSetCommand = () => {
  return new DeleteReceiptRuleSetCommand({ RuleSetName: RULE_SET_NAME });
};

const run = async () => {
  const deleteReceiptRuleSetCommand = createDeleteReceiptRuleSetCommand();

  try {
    return await sesClient.send(deleteReceiptRuleSetCommand);
  } catch (err) {
    console.log("Failed to delete receipt rule set.", err);
    return err;
  }
};
```

- Einzelheiten zur API finden Sie [DeleteReceiptRuleSet](#) in der AWS SDK für JavaScript API-Referenz.

DeleteTemplate

Das folgende Codebeispiel zeigt die Verwendung `DeleteTemplate`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { DeleteTemplateCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "../libs/sesClient.js";

const TEMPLATE_NAME = getUniqueName("TemplateName");

const createDeleteTemplateCommand = (templateName) =>
  new DeleteTemplateCommand({ TemplateName: templateName });

const run = async () => {
  const deleteTemplateCommand = createDeleteTemplateCommand(TEMPLATE_NAME);

  try {
    return await sesClient.send(deleteTemplateCommand);
  } catch (err) {
    console.log("Failed to delete template.", err);
    return err;
  }
};
```

- Einzelheiten zur API finden Sie [DeleteTemplate](#) in der AWS SDK für JavaScript API-Referenz.

GetTemplate

Das folgende Codebeispiel zeigt die Verwendung `GetTemplate`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { GetTemplateCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "../libs/sesClient.js";

const TEMPLATE_NAME = getUniqueName("TemplateName");

const createGetTemplateCommand = (templateName) =>
  new GetTemplateCommand({ TemplateName: templateName });

const run = async () => {
  const getTemplateCommand = createGetTemplateCommand(TEMPLATE_NAME);

  try {
    return await sesClient.send(getTemplateCommand);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected } */
      const messageRejectedError = caught;
      return messageRejectedError;
    }
    throw caught;
  }
};
```

- Einzelheiten zur API finden Sie [GetTemplate](#) in der AWS SDK für JavaScript API-Referenz.

ListIdentities

Das folgende Codebeispiel zeigt die Verwendung `ListIdentities`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { ListIdentitiesCommand } from "@aws-sdk/client-ses";
import { sesClient } from "../libs/sesClient.js";

const createListIdentitiesCommand = () =>
  new ListIdentitiesCommand({ IdentityType: "EmailAddress", MaxItems: 10 });

const run = async () => {
  const listIdentitiesCommand = createListIdentitiesCommand();

  try {
    return await sesClient.send(listIdentitiesCommand);
  } catch (err) {
    console.log("Failed to list identities.", err);
    return err;
  }
};
```

- Einzelheiten zur API finden Sie [ListIdentities](#) in der AWS SDK für JavaScript API-Referenz.

ListReceiptFilters

Das folgende Codebeispiel zeigt die Verwendung `ListReceiptFilters`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { ListReceiptFiltersCommand } from "@aws-sdk/client-ses";
import { sesClient } from "../libs/sesClient.js";

const createListReceiptFiltersCommand = () => new ListReceiptFiltersCommand({});

const run = async () => {
  const listReceiptFiltersCommand = createListReceiptFiltersCommand();

  return await sesClient.send(listReceiptFiltersCommand);
};
```

- Einzelheiten zur API finden Sie [ListReceiptFilters](#) in der AWS SDK für JavaScript API-Referenz.

ListTemplates

Das folgende Codebeispiel zeigt die Verwendung `ListTemplates`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { ListTemplatesCommand } from "@aws-sdk/client-ses";
import { sesClient } from "../libs/sesClient.js";

const createListTemplatesCommand = (maxItems) =>
  new ListTemplatesCommand({ MaxItems: maxItems });

const run = async () => {
  const listTemplatesCommand = createListTemplatesCommand(10);

  try {
    return await sesClient.send(listTemplatesCommand);
  } catch (err) {
    console.log("Failed to list templates.", err);
    return err;
  }
}
```

```
};
```

- Einzelheiten zur API finden Sie [ListTemplates](#) in der AWS SDK für JavaScript API-Referenz.

SendBulkTemplatedEmail

Das folgende Codebeispiel zeigt die Verwendung `SendBulkTemplatedEmail`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { SendBulkTemplatedEmailCommand } from "@aws-sdk/client-ses";
import {
  getUniqueName,
  postfix,
} from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "../libs/sesClient.js";

/**
 * Replace this with the name of an existing template.
 */
const TEMPLATE_NAME = getUniqueName("ReminderTemplate");

/**
 * Replace these with existing verified emails.
 */
const VERIFIED_EMAIL_1 = postfix(getUniqueName("Bilbo"), "@example.com");
const VERIFIED_EMAIL_2 = postfix(getUniqueName("Frodo"), "@example.com");

const USERS = [
  { firstName: "Bilbo", emailAddress: VERIFIED_EMAIL_1 },
  { firstName: "Frodo", emailAddress: VERIFIED_EMAIL_2 },
];

/**
 *
```

```

* @param { { emailAddress: string, firstName: string }[] } users
* @param { string } templateName the name of an existing template in SES
* @returns { SendBulkTemplatedEmailCommand }
*/
const createBulkReminderEmailCommand = (users, templateName) => {
  return new SendBulkTemplatedEmailCommand({
    /**
     * Each 'Destination' uses a corresponding set of replacement data. We can map
     each user
     * to a 'Destination' and provide user specific replacement data to create
     personalized emails.
     *
     * Here's an example of how a template would be replaced with user data:
     * Template: <h1>Hello {{name}},</h1><p>Don't forget about the party gifts!</p>
     * Destination 1: <h1>Hello Bilbo,</h1><p>Don't forget about the party gifts!</
p>
     * Destination 2: <h1>Hello Frodo,</h1><p>Don't forget about the party gifts!</
p>
     */
    Destinations: users.map((user) => ({
      Destination: { ToAddresses: [user.emailAddress] },
      ReplacementTemplateData: JSON.stringify({ name: user.firstName }),
    })),
    DefaultTemplateData: JSON.stringify({ name: "Shireling" }),
    Source: VERIFIED_EMAIL_1,
    Template: templateName,
  });
};

const run = async () => {
  const sendBulkTemplateEmailCommand = createBulkReminderEmailCommand(
    USERS,
    TEMPLATE_NAME,
  );
  try {
    return await sesClient.send(sendBulkTemplateEmailCommand);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected } */
      const messageRejectedError = caught;
      return messageRejectedError;
    }
    throw caught;
  }
}

```

```
};
```

- Einzelheiten zur API finden Sie [SendBulkTemplatedEmail](#) in der AWS SDK für JavaScript API-Referenz.

SendEmail

Das folgende Codebeispiel zeigt die Verwendung `SendEmail`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { SendEmailCommand } from "@aws-sdk/client-ses";
import { sesClient } from "../libs/sesClient.js";

const createSendEmailCommand = (toAddress, fromAddress) => {
  return new SendEmailCommand({
    Destination: {
      /* required */
      CcAddresses: [
        /* more items */
      ],
      ToAddresses: [
        toAddress,
        /* more To-email addresses */
      ],
    },
    Message: {
      /* required */
      Body: {
        /* required */
        Html: {
          Charset: "UTF-8",
          Data: "HTML_FORMAT_BODY",
        },
      },
    },
  });
};
```

```

        Text: {
          Charset: "UTF-8",
          Data: "TEXT_FORMAT_BODY",
        },
      },
      Subject: {
        Charset: "UTF-8",
        Data: "EMAIL_SUBJECT",
      },
    },
    Source: fromAddress,
    ReplyToAddresses: [
      /* more items */
    ],
  });
};

const run = async () => {
  const sendEmailCommand = createSendEmailCommand(
    "recipient@example.com",
    "sender@example.com",
  );

  try {
    return await sesClient.send(sendEmailCommand);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected } */
      const messageRejectedError = caught;
      return messageRejectedError;
    }
    throw caught;
  }
};

```

- Einzelheiten zur API finden Sie [SendEmail](#) in der AWS SDK für JavaScript API-Referenz.

SendRawEmail

Das folgende Codebeispiel zeigt die Verwendung `SendRawEmail`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Verwenden Sie [nodemailer](#), um eine E-Mail mit einem Anhang zu senden.

```
import sesClientModule from "@aws-sdk/client-ses";
/**
 * nodemailer wraps the SES SDK and calls SendRawEmail. Use this for more advanced
 * functionality like adding attachments to your email.
 *
 * https://nodemailer.com/transports/ses
 */
import nodemailer from "nodemailer";

/**
 * @param {string} from An Amazon SES verified email address.
 * @param {*} to An Amazon SES verified email address.
 */
export const sendEmailWithAttachments = (
  from = "from@example.com",
  to = "to@example.com",
) => {
  const ses = new sesClientModule.SESClient({});
  const transporter = nodemailer.createTransport({
    SES: { ses, aws: sesClientModule },
  });

  return new Promise((resolve, reject) => {
    transporter.sendMail(
      {
        from,
        to,
        subject: "Hello World",
        text: "Greetings from Amazon SES!",
        attachments: [{ content: "Hello World!", filename: "hello.txt" }],
      },
      (err, info) => {
        if (err) {
```



```
        reject(err);
      } else {
        resolve(info);
      }
    },
  );
});
};
```

- Einzelheiten zur API finden Sie [SendRawEmail](#) in der AWS SDK für JavaScript API-Referenz.

SendTemplatedEmail

Das folgende Codebeispiel zeigt die Verwendung `SendTemplatedEmail`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { SendTemplatedEmailCommand } from "@aws-sdk/client-ses";
import {
  getUniqueName,
  postfix,
} from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "../libs/sesClient.js";

/**
 * Replace this with the name of an existing template.
 */
const TEMPLATE_NAME = getUniqueName("ReminderTemplate");

/**
 * Replace these with existing verified emails.
 */
const VERIFIED_EMAIL = postfix(getUniqueName("Bilbo"), "@example.com");
```

```

const USER = { firstName: "Bilbo", emailAddress: VERIFIED_EMAIL };

/**
 *
 * @param { { emailAddress: string, firstName: string } } user
 * @param { string } templateName - The name of an existing template in Amazon SES.
 * @returns { SendTemplatedEmailCommand }
 */
const createReminderEmailCommand = (user, templateName) => {
  return new SendTemplatedEmailCommand({
    /**
     * Here's an example of how a template would be replaced with user data:
     * Template: <h1>Hello {{contact.firstName}},</h1><p>Don't forget about the
party gifts!</p>
     * Destination: <h1>Hello Bilbo,</h1><p>Don't forget about the party gifts!</p>
     */
    Destination: { ToAddresses: [user.emailAddress] },
    TemplateData: JSON.stringify({ contact: { firstName: user.firstName } }),
    Source: VERIFIED_EMAIL,
    Template: templateName,
  });
};

const run = async () => {
  const sendReminderEmailCommand = createReminderEmailCommand(
    USER,
    TEMPLATE_NAME,
  );
  try {
    return await sesClient.send(sendReminderEmailCommand);
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MessageRejected") {
      /** @type { import('@aws-sdk/client-ses').MessageRejected } */
      const messageRejectedError = caught;
      return messageRejectedError;
    }
    throw caught;
  }
};

```

- Einzelheiten zur API finden Sie [SendTemplatedEmail](#) in der AWS SDK für JavaScript API-Referenz.

UpdateTemplate

Das folgende Codebeispiel zeigt die Verwendung `UpdateTemplate`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { UpdateTemplateCommand } from "@aws-sdk/client-ses";
import { getUniqueName } from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "../libs/sesClient.js";

const TEMPLATE_NAME = getUniqueName("TemplateName");
const HTML_PART = "<h1>Hello, World!</h1>";

const createUpdateTemplateCommand = () => {
  return new UpdateTemplateCommand({
    Template: {
      TemplateName: TEMPLATE_NAME,
      HtmlPart: HTML_PART,
      SubjectPart: "Example",
      TextPart: "Updated template text.",
    },
  });
};

const run = async () => {
  const updateTemplateCommand = createUpdateTemplateCommand();

  try {
    return await sesClient.send(updateTemplateCommand);
  } catch (err) {
    console.log("Failed to update template.", err);
    return err;
  }
};
```

- Einzelheiten zur API finden Sie [UpdateTemplate](#) in der AWS SDK für JavaScript API-Referenz.

VerifyDomainIdentity

Das folgende Codebeispiel zeigt die Verwendung `VerifyDomainIdentity`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { VerifyDomainIdentityCommand } from "@aws-sdk/client-ses";
import {
  getUniqueName,
  postfix,
} from "@aws-doc-sdk-examples/lib/utils/util-string.js";
import { sesClient } from "../libs/sesClient.js";

/**
 * You must have access to the domain's DNS settings to complete the
 * domain verification process.
 */
const DOMAIN_NAME = postfix(getUniqueName("Domain"), ".example.com");

const createVerifyDomainIdentityCommand = () => {
  return new VerifyDomainIdentityCommand({ Domain: DOMAIN_NAME });
};

const run = async () => {
  const VerifyDomainIdentityCommand = createVerifyDomainIdentityCommand();

  try {
    return await sesClient.send(VerifyDomainIdentityCommand);
  } catch (err) {
    console.log("Failed to verify domain.", err);
    return err;
  }
};
```

- Einzelheiten zur API finden Sie [VerifyDomainIdentity](#) in der AWS SDK für JavaScript API-Referenz.

VerifyEmailIdentity

Das folgende Codebeispiel zeigt die Verwendung `VerifyEmailIdentity`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
// Import required AWS SDK clients and commands for Node.js
import { VerifyEmailIdentityCommand } from "@aws-sdk/client-ses";
import { sesClient } from "../libs/sesClient.js";

const EMAIL_ADDRESS = "name@example.com";

const createVerifyEmailIdentityCommand = (emailAddress) => {
  return new VerifyEmailIdentityCommand({ EmailAddress: emailAddress });
};

const run = async () => {
  const verifyEmailIdentityCommand =
    createVerifyEmailIdentityCommand(EMAIL_ADDRESS);
  try {
    return await sesClient.send(verifyEmailIdentityCommand);
  } catch (err) {
    console.log("Failed to verify email identity.", err);
    return err;
  }
};
```

- Einzelheiten zur API finden Sie [VerifyEmailIdentity](#) in der AWS SDK für JavaScript API-Referenz.

Szenarien

Erstellen einer Amazon-Transcribe-Streaming-App

Das folgende Code-Beispiel zeigt, wie Sie eine App erstellen, die Live-Audio in Echtzeit aufzeichnet, transkribiert und übersetzt und die Ergebnisse per E-Mail sendet.

SDK für JavaScript (v3)

Zeigt, wie Amazon Transcribe verwendet wird, um eine App zu erstellen, die Live-Audio in Echtzeit aufzeichnet, transkribiert und übersetzt und die Ergebnisse mit Amazon Simple Email Service (Amazon SES) per E-Mail sendet.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

In diesem Beispiel verwendete Dienste

- Amazon Comprehend
- Amazon SES
- Amazon Transcribe
- Amazon Translate

Erstellen eines Trackers für Aurora-Serverless-Arbeitsaufgaben

Das folgende Codebeispiel zeigt, wie eine Webanwendung erstellt wird, die Arbeitselemente in einer Datenbank von Amazon Aurora Serverless verfolgt und mithilfe von Amazon Simple Email Service (Amazon SES) Berichte sendet.

SDK für JavaScript (v3)

Zeigt, wie Sie mit AWS SDK für JavaScript (v3) eine Webanwendung erstellen, die Arbeitselemente in einer Amazon Aurora Datenbank verfolgt und Berichte mithilfe von Amazon Simple Email Service (Amazon SES) per E-Mail versendet. In diesem Beispiel wird ein mit React.js erstelltes Frontend verwendet, um mit einem Express-Node.js-Backend zu interagieren.

- Integrieren Sie eine React.js Webanwendung mit AWS-Services.
- Auflisten, hinzufügen und aktualisieren von Elementen in einer Aurora-Tabelle.

- Senden Sie einen E-Mail-Bericht über gefilterte Arbeitselemente mit Amazon SES.
- Stellen Sie Beispielressourcen mit dem mitgelieferten AWS CloudFormation Skript bereit und verwalten Sie sie.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

In diesem Beispiel verwendete Dienste

- Aurora
- Amazon RDS
- Amazon RDS Data Service
- Amazon SES

Erkennen von Objekten in Bildern

Die folgenden Codebeispiele zeigen, wie man eine App erstellt, die Amazon Rekognition verwendet, um Objekte nach Kategorien in Bildern zu erkennen.

SDK für JavaScript (v3)

Zeigt, wie Amazon Rekognition zusammen mit dem verwendet wird, um eine App AWS SDK für JavaScript zu erstellen, die Amazon Rekognition verwendet, um Objekte nach Kategorien in Bildern zu identifizieren, die sich in einem Amazon Simple Storage Service (Amazon S3) -Bucket befinden. Die App sendet dem Administrator eine E-Mail-Benachrichtigung mit den Ergebnissen über Amazon Simple Email Service (Amazon SES).

Lernen Sie Folgendes:

- Erstellen Sie mit Amazon Cognito einen nicht authentifizierten Benutzer.
- Analysieren Sie mit Amazon Rekognition Bilder für Objekte.
- Verifizieren Sie eine E-Mail-Adresse für Amazon SES.
- Senden Sie eine E-Mail-Benachrichtigung mit Amazon SES.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter. [GitHub](#)

In diesem Beispiel verwendete Dienste

- Amazon Rekognition

- Amazon S3
- Amazon SES

Amazon SNS SNS-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK für JavaScript (v3) mit Amazon SNS Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien anzeigen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb eines Service aufrufen oder mit anderen AWS-Services kombinieren.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kodex finden.

Themen

- [Erste Schritte](#)
- [Aktionen](#)
- [Szenarien](#)
- [Serverless-Beispiele](#)

Erste Schritte

Hello Amazon SNS

Das folgende Codebeispiel zeigt, wie Sie mit Amazon SNS beginnen können.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Initialisieren Sie einen SNS-Client und listen Sie Themen in Ihrem Konto auf.

```
import { SNSClient, paginateListTopics } from "@aws-sdk/client-sns";

export const helloSns = async () => {
  // The configuration object (`{}`) is required. If the region and credentials
  // are omitted, the SDK uses your local configuration if it exists.
  const client = new SNSClient({});

  // You can also use `ListTopicsCommand`, but to use that command you must
  // handle the pagination yourself. You can do that by sending the
  `ListTopicsCommand`
  // with the `NextToken` parameter from the previous request.
  const paginatedTopics = paginateListTopics({ client }, {});
  const topics = [];

  for await (const page of paginatedTopics) {
    if (page.Topics?.length) {
      topics.push(...page.Topics);
    }
  }

  const suffix = topics.length === 1 ? "" : "s";

  console.log(
    `Hello, Amazon SNS! You have ${topics.length} topic${suffix} in your account.`
  );
  console.log(topics.map((t) => ` * ${t.TopicArn}`).join("\n"));
};
```

- Einzelheiten zur API finden Sie [ListTopics](#) in der AWS SDK für JavaScript API-Referenz.

Aktionen

CheckIfPhoneNumberIsOptedOut

Das folgende Codebeispiel zeigt die Verwendung `CheckIfPhoneNumberIsOptedOut`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie den Client in einem separaten Modul und exportieren Sie ihn.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
import { CheckIfPhoneNumberIsOptedOutCommand } from "@aws-sdk/client-sns";

import { snsClient } from "../libs/snsClient.js";

export const checkIfPhoneNumberIsOptedOut = async (
  phoneNumber = "5555555555",
) => {
  const command = new CheckIfPhoneNumberIsOptedOutCommand({
    phoneNumber,
  });

  const response = await snsClient.send(command);
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '3341c28a-cdc8-5b39-a3ee-9fb0ee125732',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
```

```
// isOptedOut: false
// }
return response;
};
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [CheckIfPhoneNumbersOptedOut](#) in der AWS SDK für JavaScript API-Referenz.

ConfirmSubscription

Das folgende Codebeispiel zeigt die Verwendung `ConfirmSubscription`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie den Client in einem separaten Modul und exportieren Sie ihn.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
import { ConfirmSubscriptionCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} token - This token is sent the subscriber. Only subscribers
 *                        that are not AWS services (HTTP/S, email) need to be
 *                        confirmed.
```

```

* @param {string} topicArn - The ARN of the topic for which you wish to confirm a
subscription.
*/
export const confirmSubscription = async (
  token = "TOKEN",
  topicArn = "TOPIC_ARN",
) => {
  const response = await snsClient.send(
    // A subscription only needs to be confirmed if the endpoint type is
    // HTTP/S, email, or in another AWS account.
    new ConfirmSubscriptionCommand({
      Token: token,
      TopicArn: topicArn,
      // If this is true, the subscriber cannot unsubscribe while unauthenticated.
      AuthenticateOnUnsubscribe: "false",
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '4bb5bce9-805a-5517-8333-e1d2cf90b',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxxx:TOPIC_NAME:xxxxxxxx-
  xxxx-xxxx-xxxx-xxxxxxxxxxxxx'
  // }
  return response;
};

```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [ConfirmSubscription](#) in der AWS SDK für JavaScript API-Referenz.

CreateTopic

Das folgende Codebeispiel zeigt die Verwendung `CreateTopic`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie den Client in einem separaten Modul und exportieren Sie ihn.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
import { CreateTopicCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicName - The name of the topic to create.
 */
export const createTopic = async (topicName = "TOPIC_NAME") => {
  const response = await snsClient.send(
    new CreateTopicCommand({ Name: topicName }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '087b8ad2-4593-50c4-a496-d7e90b82cf3e',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:TOPIC_NAME'
  // }
```

```
    return response;
};
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [CreateTopic](#) in der AWS SDK für JavaScript API-Referenz.

DeleteTopic

Das folgende Codebeispiel zeigt die Verwendung `DeleteTopic`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie den Client in einem separaten Modul und exportieren Sie ihn.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
import { DeleteTopicCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic to delete.
 */
export const deleteTopic = async (topicArn = "TOPIC_ARN") => {
  const response = await snsClient.send(
    new DeleteTopicCommand({ TopicArn: topicArn }),
```

```
);  
console.log(response);  
// {  
//   '$metadata': {  
//     httpStatusCode: 200,  
//     requestId: 'a10e2886-5a8f-5114-af36-75bd39498332',  
//     extendedRequestId: undefined,  
//     cfId: undefined,  
//     attempts: 1,  
//     totalRetryDelay: 0  
//   }  
// }  
};
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [DeleteTopic](#) in der AWS SDK für JavaScript API-Referenz.

GetSMSAttributes

Das folgende Codebeispiel zeigt die Verwendung `GetSMSAttributes`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie den Client in einem separaten Modul und exportieren Sie ihn.

```
import { SNSClient } from "@aws-sdk/client-sns";  
  
// The AWS Region can be provided here using the `region` property. If you leave it  
// blank  
// the SDK will default to the region set in your AWS config.  
export const snsClient = new SNSClient({});
```

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
import { GetSMSAttributesCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

export const getSmsAttributes = async () => {
  const response = await snsClient.send(
    // If you have not modified the account-level mobile settings of SNS,
    // the DefaultSMSType is undefined. For this example, it was set to
    // Transactional.
    new GetSMSAttributesCommand({ attributes: ["DefaultSMSType"] }),
  );

  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '67ad8386-4169-58f1-bdb9-debd281d48d5',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   attributes: { DefaultSMSType: 'Transactional' }
  // }
  return response;
};
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie unter [Get SMSAttributes](#) in AWS SDK für JavaScript API-Referenz.

GetTopicAttributes

Das folgende Codebeispiel zeigt die Verwendung `GetTopicAttributes`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie den Client in einem separaten Modul und exportieren Sie ihn.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
import { GetTopicAttributesCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic to retrieve attributes for.
 */
export const getTopicAttributes = async (topicArn = "TOPIC_ARN") => {
  const response = await snsClient.send(
    new GetTopicAttributesCommand({
      TopicArn: topicArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '36b6a24e-5473-5d4e-ac32-ff72d9a73d94',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  // }
```

```
// Attributes: {
//   Policy: '{...}',
//   Owner: 'xxxxxxxxxxxxx',
//   SubscriptionsPending: '1',
//   TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxxx:mytopic',
//   TracingConfig: 'PassThrough',
//   EffectiveDeliveryPolicy: '{"http":{"defaultHealthyRetryPolicy":
{"minDelayTarget":20,"maxDelayTarget":20,"numRetries":3,"numMaxDelayRetries":0,"numNoDelayRe
{"headerContentType":"text/plain; charset=UTF-8"}}}',
//   SubscriptionsConfirmed: '0',
//   DisplayName: '',
//   SubscriptionsDeleted: '1'
// }
// }
return response;
};
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [GetTopicAttributes](#) in der AWS SDK für JavaScript API-Referenz.

ListSubscriptions

Das folgende Codebeispiel zeigt die Verwendung `ListSubscriptions`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie den Client in einem separaten Modul und exportieren Sie ihn.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
```

```
export const snsClient = new SNSClient({});
```

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
import { ListSubscriptionsByTopicCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic for which you wish to list
 * subscriptions.
 */
export const listSubscriptionsByTopic = async (topicArn = "TOPIC_ARN") => {
  const response = await snsClient.send(
    new ListSubscriptionsByTopicCommand({ TopicArn: topicArn }),
  );

  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '0934fedf-0c4b-572e-9ed2-a3e38fadb0c8',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Subscriptions: [
  //     {
  //       SubscriptionArn: 'PendingConfirmation',
  //       Owner: '901487484989',
  //       Protocol: 'email',
  //       Endpoint: 'corepyle@amazon.com',
  //       TopicArn: 'arn:aws:sns:us-east-1:901487484989:mytopic'
  //     }
  //   ]
  // }
  return response;
};
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [ListSubscriptions](#) in der AWS SDK für JavaScript API-Referenz.

ListTopics

Das folgende Codebeispiel zeigt die Verwendung `ListTopics`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie den Client in einem separaten Modul und exportieren Sie ihn.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
import { ListTopicsCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

export const listTopics = async () => {
  const response = await snsClient.send(new ListTopicsCommand({}));
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '936bc5ad-83ca-53c2-b0b7-9891167b909e',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   Topics: [ { TopicArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:mytopic' } ]
  // }
  return response;
}
```

```
};
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [ListTopics](#) in der AWS SDK für JavaScript API-Referenz.

Publish

Das folgende Codebeispiel zeigt die Verwendung `Publish`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie den Client in einem separaten Modul und exportieren Sie ihn.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
import { PublishCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string | Record<string, any>} message - The message to send. Can be a
plain string or an object
 *
 * if you are using the `json`
 * `MessageStructure`.
 * @param {string} topicArn - The ARN of the topic to which you would like to
publish.
```

```

*/
export const publish = async (
  message = "Hello from SNS!",
  topicArn = "TOPIC_ARN",
) => {
  const response = await snsClient.send(
    new PublishCommand({
      Message: message,
      TopicArn: topicArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'e7f77526-e295-5325-9ee4-281a43ad1f05',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   MessageId: 'xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxxx'
  // }
  return response;
};

```

Veröffentlichen Sie eine Nachricht zu einem Thema mit Gruppen-, Duplizierungs- und Attributoptionen.

```

async publishMessages() {
  const message = await this.prompter.input({
    message: MESSAGES.publishMessagePrompt,
  });

  let groupId;
  let deduplicationId;
  let choices;

  if (this.isFifo) {
    await this.logger.log(MESSAGES.groupIdNotice);
    groupId = await this.prompter.input({
      message: MESSAGES.groupIdPrompt,
    });
  }
}

```

```
});

if (this.autoDedup === false) {
  await this.logger.log(MESSAGES.deduplicationIdNotice);
  deduplicationId = await this.prompter.input({
    message: MESSAGES.deduplicationIdPrompt,
  });
}

choices = await this.prompter.checkbox({
  message: MESSAGES.messageAttributesPrompt,
  choices: toneChoices,
});
}

await this.snsClient.send(
  new PublishCommand({
    TopicArn: this.topicArn,
    Message: message,
    ...(groupId
      ? {
          MessageGroupId: groupId,
        }
      : {}),
    ...(deduplicationId
      ? {
          MessageDeduplicationId: deduplicationId,
        }
      : {}),
    ...(choices
      ? {
          MessageAttributes: {
            tone: {
              DataType: "String.Array",
              StringValue: JSON.stringify(choices),
            },
          },
        }
      : {}),
  })),
);

const publishAnother = await this.prompter.confirm({
  message: MESSAGES.publishAnother,
```

```
});  
  
if (publishAnother) {  
  await this.publishMessages();  
}  
}
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Weitere API-Informationen finden Sie unter [Publish](#) in der AWS SDK für JavaScript -API-Referenz.

SetSMSAttributes

Das folgende Codebeispiel zeigt die Verwendung `SetSMSAttributes`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel](#) einrichten und ausführen.

Erstellen Sie den Client in einem separaten Modul und exportieren Sie ihn.

```
import { SNSClient } from "@aws-sdk/client-sns";  
  
// The AWS Region can be provided here using the `region` property. If you leave it  
// blank  
// the SDK will default to the region set in your AWS config.  
export const snsClient = new SNSClient({});
```

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
import { SetSMSAttributesCommand } from "@aws-sdk/client-sns";  
import { snsClient } from "../libs/snsClient.js";  
  
/**  
 * @param {"Transactional" | "Promotional"} defaultSmsType
```



```
*/
export const setSmsType = async (defaultSmsType = "Transactional") => {
  const response = await snsClient.send(
    new SetSMSAttributesCommand({
      attributes: {
        // Promotional - (Default) Noncritical messages, such as marketing messages.
        // Transactional - Critical messages that support customer transactions,
        // such as one-time passcodes for multi-factor authentication.
        DefaultSMSType: defaultSmsType,
      },
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '1885b977-2d7e-535e-8214-e44be727e265',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
  return response;
};
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie unter [Set SMSAttributes](#) in der AWS SDK für JavaScript API-Referenz.

SetTopicAttributes

Das folgende Codebeispiel zeigt die Verwendung `SetTopicAttributes`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie den Client in einem separaten Modul und exportieren Sie ihn.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
import { SetTopicAttributesCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

export const setTopicAttributes = async (
  topicArn = "TOPIC_ARN",
  attributeName = "DisplayName",
  attributeValue = "Test Topic",
) => {
  const response = await snsClient.send(
    new SetTopicAttributesCommand({
      AttributeName: attributeName,
      AttributeValue: attributeValue,
      TopicArn: topicArn,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'd1b08d0e-e9a4-54c3-b8b1-d03238d2b935',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   }
  // }
  return response;
};
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).

- Einzelheiten zur API finden Sie [SetTopicAttributes](#) in der AWS SDK für JavaScript API-Referenz.

Subscribe

Das folgende Codebeispiel zeigt die Verwendung `Subscribe`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie den Client in einem separaten Modul und exportieren Sie ihn.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
import { SubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic for which you wish to confirm a
 * subscription.
 * @param {string} emailAddress - The email address that is subscribed to the topic.
 */
export const subscribeEmail = async (
  topicArn = "TOPIC_ARN",
  emailAddress = "usern@me.com",
) => {
  const response = await snsClient.send(
    new SubscribeCommand({
```

```

        Protocol: "email",
        TopicArn: topicArn,
        Endpoint: emailAddress,
    })),
);
console.log(response);
// {
//   '$metadata': {
//     httpStatusCode: 200,
//     requestId: 'c8e35bcd-b3c0-5940-9f66-06f6fcc108f0',
//     extendedRequestId: undefined,
//     cfId: undefined,
//     attempts: 1,
//     totalRetryDelay: 0
//   },
//   SubscriptionArn: 'pending confirmation'
// }
};

```

Abonnieren Sie eine mobile Anwendung für ein Thema.

```

import { SubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic the subscriber is subscribing to.
 * @param {string} endpoint - The Endpoint ARN of an application. This endpoint is
 * created
 *
 *                               when an application registers for notifications.
 */
export const subscribeApp = async (
  topicArn = "TOPIC_ARN",
  endpoint = "ENDPOINT",
) => {
  const response = await snsClient.send(
    new SubscribeCommand({
      Protocol: "application",
      TopicArn: topicArn,
      Endpoint: endpoint,
    }),
  );
  console.log(response);
}

```

```
// {
//   '$metadata': {
//     httpStatusCode: 200,
//     requestId: 'c8e35bcd-b3c0-5940-9f66-06f6fcc108f0',
//     extendedRequestId: undefined,
//     cfId: undefined,
//     attempts: 1,
//     totalRetryDelay: 0
//   },
//   SubscriptionArn: 'pending confirmation'
// }
return response;
};
```

Abonnieren Sie eine Lambda-Funktion für ein Thema.

```
import { SubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} topicArn - The ARN of the topic the subscriber is subscribing to.
 * @param {string} endpoint - The Endpoint ARN of and AWS Lambda function.
 */
export const subscribeLambda = async (
  topicArn = "TOPIC_ARN",
  endpoint = "ENDPOINT",
) => {
  const response = await snsClient.send(
    new SubscribeCommand({
      Protocol: "lambda",
      TopicArn: topicArn,
      Endpoint: endpoint,
    }),
  );
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: 'c8e35bcd-b3c0-5940-9f66-06f6fcc108f0',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
```

```
//      totalRetryDelay: 0
//    },
//    SubscriptionArn: 'pending confirmation'
//  }
  return response;
};
```

Abonnieren Sie eine SQS-Warteschlange für ein Thema.

```
import { SubscribeCommand, SNSClient } from "@aws-sdk/client-sns";

const client = new SNSClient({});

export const subscribeQueue = async (
  topicArn = "TOPIC_ARN",
  queueArn = "QUEUE_ARN",
) => {
  const command = new SubscribeCommand({
    TopicArn: topicArn,
    Protocol: "sqs",
    Endpoint: queueArn,
  });

  const response = await client.send(command);
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '931e13d9-5e2b-543f-8781-4e9e494c5ff2',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:subscribe-queue-
test-430895:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxx'
  // }
  return response;
};
```

Abonnieren Sie ein Thema mit einem Filter.

```
import { SubscribeCommand, SNSClient } from "@aws-sdk/client-sns";

const client = new SNSClient({});

export const subscribeQueueFiltered = async (
  topicArn = "TOPIC_ARN",
  queueArn = "QUEUE_ARN",
) => {
  const command = new SubscribeCommand({
    TopicArn: topicArn,
    Protocol: "sqs",
    Endpoint: queueArn,
    Attributes: {
      // This subscription will only receive messages with the 'event' attribute set
      // to 'order_placed'.
      FilterPolicyScope: "MessageAttributes",
      FilterPolicy: JSON.stringify({
        event: ["order_placed"],
      }),
    },
  });

  const response = await client.send(command);
  console.log(response);
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '931e13d9-5e2b-543f-8781-4e9e494c5ff2',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   SubscriptionArn: 'arn:aws:sns:us-east-1:xxxxxxxxxxxx:subscribe-queue-
  // test-430895:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxxx'
  // }
  return response;
};
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).

- Weitere API-Informationen finden Sie unter [Subscribe](#) in der AWS SDK für JavaScript -API-Referenz.

Unsubscribe

Das folgende Codebeispiel zeigt die Verwendung `Unsubscribe`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie den Client in einem separaten Modul und exportieren Sie ihn.

```
import { SNSClient } from "@aws-sdk/client-sns";

// The AWS Region can be provided here using the `region` property. If you leave it
// blank
// the SDK will default to the region set in your AWS config.
export const snsClient = new SNSClient({});
```

Importieren Sie das SDK- und Client-Module und rufen Sie die API auf.

```
import { UnsubscribeCommand } from "@aws-sdk/client-sns";
import { snsClient } from "../libs/snsClient.js";

/**
 * @param {string} subscriptionArn - The ARN of the subscription to cancel.
 */
const unsubscribe = async (
  subscriptionArn = "arn:aws:sns:us-east-1:xxxxxxxxxxxx:mytopic:xxxxxxxx-xxxx-xxxx-
  xxxx-xxxxxxxxxxxx",
) => {
  const response = await snsClient.send(
    new UnsubscribeCommand({
      SubscriptionArn: subscriptionArn,
    }),
  ),
```



```
);  
console.log(response);  
// {  
//   '$metadata': {  
//     httpStatusCode: 200,  
//     requestId: '0178259a-9204-507c-b620-78a7570a44c6',  
//     extendedRequestId: undefined,  
//     cfId: undefined,  
//     attempts: 1,  
//     totalRetryDelay: 0  
//   }  
// }  
return response;  
};
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Weitere API-Informationen finden Sie unter [Unsubscribe](#) in der AWS SDK für JavaScript -API-Referenz.

Szenarien

Erstellen Sie eine App zum Senden von Daten an eine DynamoDB-Tabelle

Die folgenden Codebeispiele zeigen, wie eine Anwendung erstellt wird, die Daten an eine Amazon-DynamoDB-Tabelle übermittelt und Sie benachrichtigt, wenn ein Benutzer die Tabelle aktualisiert.

SDK für JavaScript (v3)

Das Beispiel zeigt, wie man eine App erstellt, die es Benutzern ermöglicht, Daten an eine Amazon-DynamoDB-Tabelle zu übermitteln und eine Textnachricht an den Administrator mit Amazon Simple Notification Service (Amazon SNS) zu senden.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

Dieses Beispiel ist auch verfügbar im [AWS SDK für JavaScript Entwicklerhandbuch für v3](#).

In diesem Beispiel verwendete Dienste

- DynamoDB

- Amazon SNS

Erstellen einer Serverless-Anwendung zur Verwaltung von Fotos

Das folgende Codebeispiel zeigt, wie eine Serverless-Anwendung erstellt wird, mit der Benutzer Fotos mithilfe von Labels erstellen können.

SDK für JavaScript (v3)

Zeigt, wie eine Anwendung zur Verwaltung von Fotobeständen entwickelt wird, die mithilfe von Amazon Rekognition Labels in Bildern erkennt und sie für einen späteren Abruf speichert.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

Einen tiefen Einblick in den Ursprung dieses Beispiels finden Sie im Beitrag in der [AWS - Community](#).

In diesem Beispiel verwendete Dienste

- API Gateway
- DynamoDB
- Lambda
- Amazon Rekognition
- Amazon S3
- Amazon SNS

Erstellen Sie eine Amazon-Texttract-Explorer-Anwendung

Die folgenden Codebeispiele zeigen, wie die Amazon-Texttract-Ausgabe in einer interaktiven Anwendung untersucht werden kann.

SDK für JavaScript (v3)

Zeigt, wie Sie mit AWS SDK für JavaScript eine React-Anwendung erstellen, die Amazon Texttract verwendet, um Daten aus einem Dokumentbild zu extrahieren und auf einer interaktiven Webseite anzuzeigen. Dieses Beispiel wird in einem Webbrowser ausgeführt und erfordert eine authentifizierte Amazon-Cognito-Identität für Anmeldeinformationen. Es verwendet Amazon Simple Storage Service (Amazon S3) zur Speicherung und fragt für Benachrichtigungen eine

Amazon Simple Queue Service (Amazon SQS)-Warteschlange ab, die ein Amazon Simple Notification Service (Amazon SNS)-Thema abonniert hat.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

In diesem Beispiel verwendete Dienste

- Amazon Cognito Identity
- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon Textract

Veröffentlichen Sie Nachrichten in Warteschlangen

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Erstellen Sie ein Thema (FIFO oder Nicht-FIFO).
- Abonnieren Sie mehrere Warteschlangen für das Thema mit der Option, einen Filter anzuwenden.
- Veröffentlichen Sie eine Nachricht im Thema.
- Fragen Sie die Warteschlangen nach empfangenen Nachrichten ab.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Dies ist der Einstiegspunkt für dieses Szenario.

```
import { SNSClient } from "@aws-sdk/client-sns";
import { SQSClient } from "@aws-sdk/client-sqs";

import { TopicsQueuesWkflw } from "../TopicsQueuesWkflw.js";
import { Prompter } from "@aws-doc-sdk-examples/lib/prompter.js";
```

```
export const startSnsWorkflow = () => {
  const snsClient = new SNSClient({});
  const sqsClient = new SQSClient({});
  const prompter = new Prompter();
  const logger = console;

  const wkflw = new TopicsQueuesWkflw(snsClient, sqsClient, prompter, logger);

  wkflw.start();
};
```

Der obige Code stellt die erforderlichen Abhängigkeiten bereit und startet das Szenario. Der nächste Abschnitt enthält den Großteil des Beispiels.

```
const toneChoices = [
  { name: "cheerful", value: "cheerful" },
  { name: "funny", value: "funny" },
  { name: "serious", value: "serious" },
  { name: "sincere", value: "sincere" },
];

export class TopicsQueuesWkflw {
  // SNS topic is configured as First-In-First-Out
  isFifo = true;

  // Automatic content-based deduplication is enabled.
  autoDedup = false;

  snsClient;
  sqsClient;
  topicName;
  topicArn;
  subscriptionArns = [];
  /**
   * @type {{ queueName: string, queueArn: string, queueUrl: string, policy?:
string }[]}
   */
  queues = [];
  prompter;
```

```
/**
 * @param {import('@aws-sdk/client-sns').SNSClient} snsClient
 * @param {import('@aws-sdk/client-sqs').SQSClient} sqsClient
 * @param {import('../libs/prompter.js').Prompter} prompter
 * @param {import('../libs/logger.js').Logger} logger
 */
constructor(snsClient, sqsClient, prompter, logger) {
  this.snsClient = snsClient;
  this.sqsClient = sqsClient;
  this.prompter = prompter;
  this.logger = logger;
}

async welcome() {
  await this.logger.log(MESSAGES.description);
}

async confirmFifo() {
  await this.logger.log(MESSAGES.snsFifoDescription);
  this.isFifo = await this.prompter.confirm({
    message: MESSAGES.snsFifoPrompt,
  });

  if (this.isFifo) {
    this.logger.logSeparator(MESSAGES.headerDedup);
    await this.logger.log(MESSAGES.deduplicationNotice);
    await this.logger.log(MESSAGES.deduplicationDescription);
    this.autoDedup = await this.prompter.confirm({
      message: MESSAGES.deduplicationPrompt,
    });
  }
}

async createTopic() {
  await this.logger.log(MESSAGES.creatingTopics);
  this.topicName = await this.prompter.input({
    message: MESSAGES.topicNamePrompt,
  });
  if (this.isFifo) {
    this.topicName += ".fifo";
    this.logger.logSeparator(MESSAGES.headerFifoNaming);
    await this.logger.log(MESSAGES.appendFifoNotice);
  }
}
```

```
const response = await this.snsClient.send(
  new CreateTopicCommand({
    Name: this.topicName,
    Attributes: {
      FifoTopic: this.isFifo ? "true" : "false",
      ...(this.autoDedup ? { ContentBasedDeduplication: "true" } : {}),
    },
  }),
);

this.topicArn = response.TopicArn;

await this.logger.log(
  MESSAGES.topicCreatedNotice
    .replace("${TOPIC_NAME}", this.topicName)
    .replace("${TOPIC_ARN}", this.topicArn),
);
}

async createQueues() {
  await this.logger.log(MESSAGES.createQueuesNotice);
  // Increase this number to add more queues.
  const maxQueues = 2;

  for (let i = 0; i < maxQueues; i++) {
    await this.logger.log(MESSAGES.queueCount.replace("${COUNT}", i + 1));
    let queueName = await this.prompter.input({
      message: MESSAGES.queueNamePrompt.replace(
        "${EXAMPLE_NAME}",
        i === 0 ? "good-news" : "bad-news",
      ),
    });

    if (this.isFifo) {
      queueName += ".fifo";
      await this.logger.log(MESSAGES.appendFifoNotice);
    }

    const response = await this.sqsClient.send(
      new CreateQueueCommand({
        QueueName: queueName,
        Attributes: { ...(this.isFifo ? { FifoQueue: "true" } : {}) },
      }),
    );
  }
}
```

```
);

const { Attributes } = await this.sqsClient.send(
  new GetQueueAttributesCommand({
    QueueUrl: response.QueueUrl,
    AttributeNames: ["QueueArn"],
  }),
);

this.queues.push({
  queueName,
  queueArn: Attributes.QueueArn,
  queueUrl: response.QueueUrl,
});

await this.logger.log(
  MESSAGES.queueCreatedNotice
    .replace("${QUEUE_NAME}", queueName)
    .replace("${QUEUE_URL}", response.QueueUrl)
    .replace("${QUEUE_ARN}", Attributes.QueueArn),
);
}
}

async attachQueueIamPolicies() {
  for (const [index, queue] of this.queues.entries()) {
    const policy = JSON.stringify(
      {
        Statement: [
          {
            Effect: "Allow",
            Principal: {
              Service: "sns.amazonaws.com",
            },
            Action: "sqs:SendMessage",
            Resource: queue.queueArn,
            Condition: {
              ArnEquals: {
                "aws:SourceArn": this.topicArn,
              },
            },
          },
        ],
      },
    );
  },
}
```

```

        null,
        2,
    );

    if (index !== 0) {
        this.logger.logSeparator();
    }

    await this.logger.log(MESSAGES.attachPolicyNotice);
    console.log(policy);
    const addPolicy = await this.prompter.confirm({
        message: MESSAGES.addPolicyConfirmation.replace(
            "${QUEUE_NAME}",
            queue.queueName,
        ),
    });

    if (addPolicy) {
        await this.sqsClient.send(
            new SetQueueAttributesCommand({
                QueueUrl: queue.queueUrl,
                Attributes: {
                    Policy: policy,
                },
            }),
        );
        queue.policy = policy;
    } else {
        await this.logger.log(
            MESSAGES.policyNotAttachedNotice.replace(
                "${QUEUE_NAME}",
                queue.queueName,
            ),
        );
    }
}

async subscribeQueuesToTopic() {
    for (const [index, queue] of this.queues.entries()) {
        /**
         * @type {import('@aws-sdk/client-sns').SubscribeCommandInput}
         */
        const subscribeParams = {

```



```

        TopicArn: this.topicArn,
        Protocol: "sqs",
        Endpoint: queue.queueArn,
    };
    let tones = [];

    if (this.isFifo) {
        if (index === 0) {
            await this.logger.log(MESSAGES.fifoFilterNotice);
        }
        tones = await this.prompter.checkbox({
            message: MESSAGES.fifoFilterSelect.replace(
                "${QUEUE_NAME}",
                queue.queueName,
            ),
            choices: toneChoices,
        });

        if (tones.length) {
            subscribeParams.Attributes = {
                FilterPolicyScope: "MessageAttributes",
                FilterPolicy: JSON.stringify({
                    tone: tones,
                }),
            };
        }
    }

    const { SubscriptionArn } = await this.snsClient.send(
        new SubscribeCommand(subscribeParams),
    );

    this.subscriptionArns.push(SubscriptionArn);

    await this.logger.log(
        MESSAGES.queueSubscribedNotice
            .replace("${QUEUE_NAME}", queue.queueName)
            .replace("${TOPIC_NAME}", this.topicName)
            .replace("${TONES}", tones.length ? tones.join(", ") : "none"),
    );
}
}

async publishMessages() {

```

```
const message = await this.prompter.input({
  message: MESSAGES.publishMessagePrompt,
});

let groupId;
let deduplicationId;
let choices;

if (this.isFifo) {
  await this.logger.log(MESSAGES.groupIdNotice);
  groupId = await this.prompter.input({
    message: MESSAGES.groupIdPrompt,
  });

  if (this.autoDedup === false) {
    await this.logger.log(MESSAGES.deduplicationIdNotice);
    deduplicationId = await this.prompter.input({
      message: MESSAGES.deduplicationIdPrompt,
    });
  }

  choices = await this.prompter.checkbox({
    message: MESSAGES.messageAttributesPrompt,
    choices: toneChoices,
  });
}

await this.snsClient.send(
  new PublishCommand({
    TopicArn: this.topicArn,
    Message: message,
    ...(groupId
      ? {
          MessageGroupId: groupId,
        }
      : {}),
    ...(deduplicationId
      ? {
          MessageDeduplicationId: deduplicationId,
        }
      : {}),
    ...(choices
      ? {
          MessageAttributes: {
```

```
        tone: {
          DataType: "String.Array",
          StringValue: JSON.stringify(choices),
        },
      },
    },
    : {}),
  })),
);

const publishAnother = await this.prompter.confirm({
  message: MESSAGES.publishAnother,
});

if (publishAnother) {
  await this.publishMessages();
}
}

async receiveAndDeleteMessages() {
  for (const queue of this.queues) {
    const { Messages } = await this.sqsClient.send(
      new ReceiveMessageCommand({
        QueueUrl: queue.queueUrl,
      }),
    );

    if (Messages) {
      await this.logger.log(
        MESSAGES.messagesReceivedNotice.replace(
          "${QUEUE_NAME}",
          queue.queueName,
        ),
      );
      console.log(Messages);

      await this.sqsClient.send(
        new DeleteMessageBatchCommand({
          QueueUrl: queue.queueUrl,
          Entries: Messages.map((message) => ({
            Id: message.MessageId,
            ReceiptHandle: message.ReceiptHandle,
          })),
        }),
      );
    }
  }
}
```

```
    );
  } else {
    await this.logger.log(
      MESSAGES.noMessagesReceivedNotice.replace(
        "${QUEUE_NAME}",
        queue.queueName,
      ),
    );
  }
}

const deleteAndPoll = await this.prompter.confirm({
  message: MESSAGES.deleteAndPollConfirmation,
});

if (deleteAndPoll) {
  await this.receiveAndDeleteMessages();
}

async destroyResources() {
  for (const subscriptionArn of this.subscriptionArns) {
    await this.snsClient.send(
      new UnsubscribeCommand({ SubscriptionArn: subscriptionArn }),
    );
  }

  for (const queue of this.queues) {
    await this.sqsClient.send(
      new DeleteQueueCommand({ QueueUrl: queue.queueUrl }),
    );
  }

  if (this.topicArn) {
    await this.snsClient.send(
      new DeleteTopicCommand({ TopicArn: this.topicArn }),
    );
  }
}

async start() {
  console.clear();

  try {
```

```
    this.logger.logSeparator(MESSAGES.headerWelcome);
    await this.welcome();
    this.logger.logSeparator(MESSAGES.headerFifo);
    await this.confirmFifo();
    this.logger.logSeparator(MESSAGES.headerCreateTopic);
    await this.createTopic();
    this.logger.logSeparator(MESSAGES.headerCreateQueues);
    await this.createQueues();
    this.logger.logSeparator(MESSAGES.headerAttachPolicy);
    await this.attachQueueIamPolicies();
    this.logger.logSeparator(MESSAGES.headerSubscribeQueues);
    await this.subscribeQueuesToTopic();
    this.logger.logSeparator(MESSAGES.headerPublishMessage);
    await this.publishMessages();
    this.logger.logSeparator(MESSAGES.headerReceiveMessages);
    await this.receiveAndDeleteMessages();
  } catch (err) {
    console.error(err);
  } finally {
    await this.destroyResources();
  }
}
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für JavaScript - API-Referenz.
 - [CreateQueue](#)
 - [CreateTopic](#)
 - [DeleteMessageBatch](#)
 - [DeleteQueue](#)
 - [DeleteTopic](#)
 - [GetQueueAttributes](#)
 - [Veröffentlichen](#)
 - [ReceiveMessage](#)
 - [SetQueueAttributes](#)
 - [Abonnieren](#)
 - [Unsubscribe](#)

Verwenden von API Gateway zum Aufrufen einer Lambda-Funktion

Das folgende Codebeispiel zeigt, wie eine von Amazon API Gateway aufgerufene AWS Lambda Funktion erstellt wird.

SDK für JavaScript (v3)

Zeigt, wie eine AWS Lambda Funktion mithilfe der JavaScript Lambda-Laufzeit-API erstellt wird. In diesem Beispiel werden verschiedene AWS Dienste aufgerufen, um einen bestimmten Anwendungsfall auszuführen. Dieses Beispiel zeigt, wie man eine Lambda-Funktion erstellt, die von Amazon API Gateway aufgerufen wird und eine Amazon-DynamoDB-Tabelle nach Arbeitsjubiläen durchsucht und Amazon Simple Notification Service (Amazon SNS) verwendet, um eine Textnachricht an Ihre Mitarbeiter zu senden, die ihnen zu ihrem einjährigen Jubiläum gratuliert.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

Dieses Beispiel ist auch verfügbar im [AWS SDK für JavaScript Entwicklerhandbuch für v3](#).

In diesem Beispiel verwendete Dienste

- API Gateway
- DynamoDB
- Lambda
- Amazon SNS

Verwendung geplanter Ereignisse zum Aufrufen einer Lambda-Funktion

Das folgende Codebeispiel zeigt, wie eine AWS Lambda Funktion erstellt wird, die durch ein von Amazon EventBridge geplantes Ereignis aufgerufen wird.

SDK für JavaScript (v3)

Zeigt, wie ein von Amazon EventBridge geplantes Ereignis erstellt wird, das eine AWS Lambda Funktion aufruft. Konfigurieren Sie so EventBridge, dass ein Cron-Ausdruck verwendet wird, um zu planen, wann die Lambda-Funktion aufgerufen wird. In diesem Beispiel erstellen Sie eine Lambda-Funktion mithilfe der JavaScript Lambda-Laufzeit-API. In diesem Beispiel werden verschiedene AWS Dienste aufgerufen, um einen bestimmten Anwendungsfall auszuführen.

Dieses Beispiel zeigt, wie man eine App erstellt, die eine mobile Textnachricht an Ihre Mitarbeiter sendet, um ihnen zum einjährigen Jubiläum zu gratulieren.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

Dieses Beispiel ist auch verfügbar im [AWS SDK für JavaScript Entwicklerhandbuch für v3](#).

In diesem Beispiel verwendete Dienste

- CloudWatch Logs
- DynamoDB
- EventBridge
- Lambda
- Amazon SNS

Serverless-Beispiele

Eine Lambda-Funktion über einen Amazon-SNS-Trigger aufrufen

Im folgenden Codebeispiel wird die Implementierung einer Lambda-Funktion veranschaulicht, die ein Ereignis empfängt, das durch das Empfangen von Nachrichten aus einem SNS-Thema ausgelöst wird. Die Funktion ruft die Nachrichten aus dem Ereignisparameter ab und protokolliert den Inhalt jeder Nachricht.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Konsumieren eines SNS-Ereignisses mit Lambda unter Verwendung. JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const record of event.Records) {
    await processMessageAsync(record);
  }
}
```

```

    }
    console.info("done");
  };

  async function processMessageAsync(record) {
    try {
      const message = JSON.stringify(record.Sns.Message);
      console.log(`Processed message ${message}`);
      await Promise.resolve(1); //Placeholder for actual async work
    } catch (err) {
      console.error("An error occurred");
      throw err;
    }
  }
}

```

Konsumieren eines SNS-Ereignisses mit Lambda unter Verwendung. TypeScript

```

// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { SNSEvent, Context, SNSHandler, SNSEventRecord } from "aws-lambda";

export const functionHandler: SNSHandler = async (
  event: SNSEvent,
  context: Context
): Promise<void> => {
  for (const record of event.Records) {
    await processMessageAsync(record);
  }
  console.info("done");
};

async function processMessageAsync(record: SNSEventRecord): Promise<any> {
  try {
    const message: string = JSON.stringify(record.Sns.Message);
    console.log(`Processed message ${message}`);
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}

```


Amazon SQS SQS-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK für JavaScript (v3) mit Amazon SQS Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien anzeigen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb eines Service aufrufen oder mit anderen AWS-Services kombinieren.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kodex finden.

Themen

- [Erste Schritte](#)
- [Aktionen](#)
- [Szenarien](#)
- [Serverless-Beispiele](#)

Erste Schritte

Hello Amazon SQS

Das folgende Codebeispiel zeigt die ersten Schritte mit Amazon SQS.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Initialisieren Sie einen Amazon-SQS-Client und listen Sie Warteschlangen auf.

```
import { SQSClient, paginateListQueues } from "@aws-sdk/client-sqs";
```

```
export const helloSqs = async () => {
  // The configuration object ( `{}` ) is required. If the region and credentials
  // are omitted, the SDK uses your local configuration if it exists.
  const client = new SQSClient({});

  // You can also use `ListQueuesCommand`, but to use that command you must
  // handle the pagination yourself. You can do that by sending the
  `ListQueuesCommand`
  // with the `NextToken` parameter from the previous request.
  const paginatedQueues = paginateListQueues({ client }, {});
  const queues = [];

  for await (const page of paginatedQueues) {
    if (page.QueueUrls?.length) {
      queues.push(...page.QueueUrls);
    }
  }

  const suffix = queues.length === 1 ? "" : "s";

  console.log(
    `Hello, Amazon SQS! You have ${queues.length} queue${suffix} in your account.`,
  );
  console.log(queues.map((t) => ` * ${t}`).join("\n"));
};
```

- Einzelheiten zur API finden Sie [ListQueues](#) in der AWS SDK für JavaScript API-Referenz.

Aktionen

ChangeMessageVisibility

Das folgende Codebeispiel zeigt die Verwendung `ChangeMessageVisibility`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Empfangen Sie eine Amazon-SQS-Nachricht und ändern Sie deren Timeout-Sichtbarkeit.

```
import {
  ReceiveMessageCommand,
  ChangeMessageVisibilityCommand,
  SQSClient,
} from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

const receiveMessage = (queueUrl) =>
  client.send(
    new ReceiveMessageCommand({
      AttributeNames: ["SentTimestamp"],
      MaxNumberOfMessages: 1,
      MessageAttributeNames: ["All"],
      QueueUrl: queueUrl,
      WaitTimeSeconds: 1,
    }),
  );

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const { Messages } = await receiveMessage(queueUrl);

  const response = await client.send(
    new ChangeMessageVisibilityCommand({
      QueueUrl: queueUrl,
      ReceiptHandle: Messages[0].ReceiptHandle,
      VisibilityTimeout: 20,
    }),
  );
  console.log(response);
  return response;
};
```

- Einzelheiten zur API finden Sie [ChangeMessageVisibility](#) in der AWS SDK für JavaScript API-Referenz.

CreateQueue

Das folgende Codebeispiel zeigt die Verwendung `CreateQueue`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie eine Standard-Warteschlange von Amazon SQS.

```
import { CreateQueueCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_NAME = "test-queue";

export const main = async (sqsQueueName = SQS_QUEUE_NAME) => {
  const command = new CreateQueueCommand({
    QueueName: sqsQueueName,
    Attributes: {
      DelaySeconds: "60",
      MessageRetentionPeriod: "86400",
    },
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

Erstellen einer Amazon-SQS-Warteschlange mit Langabfrage

```
import { CreateQueueCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_NAME = "queue_name";

export const main = async (queueName = SQS_QUEUE_NAME) => {
```

```
const response = await client.send(  
  new CreateQueueCommand({  
    QueueName: queueName,  
    Attributes: {  
      // When the wait time for the ReceiveMessage API action is greater than 0,  
      // long polling is in effect. The maximum long polling wait time is 20  
      // seconds. Long polling helps reduce the cost of using Amazon SQS by,  
      // eliminating the number of empty responses and false empty responses.  
      // https://docs.aws.amazon.com/AWSSimpleQueueService/latest/  
      SQSDeveloperGuide/sqs-short-and-long-polling.html  
      ReceiveMessageWaitTimeSeconds: "20",  
    },  
  }  
);  
console.log(response);  
return response;  
};
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [CreateQueue](#) in der AWS SDK für JavaScript API-Referenz.

DeleteMessage

Das folgende Codebeispiel zeigt die Verwendung `DeleteMessage`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Rufen Sie Amazon-SQS-Nachrichten ab und löschen Sie sie.

```
import {  
  ReceiveMessageCommand,  
  DeleteMessageCommand,  
  SQSClient,  
  DeleteMessageBatchCommand,  
} from "@aws-sdk/client-sqs";
```

```
const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

const receiveMessage = (queueUrl) =>
  client.send(
    new ReceiveMessageCommand({
      AttributeNames: ["SentTimestamp"],
      MaxNumberOfMessages: 10,
      MessageAttributeNames: ["All"],
      QueueUrl: queueUrl,
      WaitTimeSeconds: 20,
      VisibilityTimeout: 20,
    }),
  );

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const { Messages } = await receiveMessage(queueUrl);

  if (!Messages) {
    return;
  }

  if (Messages.length === 1) {
    console.log(Messages[0].Body);
    await client.send(
      new DeleteMessageCommand({
        QueueUrl: queueUrl,
        ReceiptHandle: Messages[0].ReceiptHandle,
      }),
    );
  } else {
    await client.send(
      new DeleteMessageBatchCommand({
        QueueUrl: queueUrl,
        Entries: Messages.map((message) => ({
          Id: message.MessageId,
          ReceiptHandle: message.ReceiptHandle,
        })),
      }),
    );
  }
};
```

- Einzelheiten zur API finden Sie [DeleteMessage](#) in der AWS SDK für JavaScript API-Referenz.

DeleteMessageBatch

Das folgende Codebeispiel zeigt die Verwendung `DeleteMessageBatch`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import {
  ReceiveMessageCommand,
  DeleteMessageCommand,
  SQSClient,
  DeleteMessageBatchCommand,
} from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

const receiveMessage = (queueUrl) =>
  client.send(
    new ReceiveMessageCommand({
      AttributeNames: ["SentTimestamp"],
      MaxNumberOfMessages: 10,
      MessageAttributeNames: ["All"],
      QueueUrl: queueUrl,
      WaitTimeSeconds: 20,
      VisibilityTimeout: 20,
    })
  );

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const { Messages } = await receiveMessage(queueUrl);
```

```
if (!Messages) {
  return;
}

if (Messages.length === 1) {
  console.log(Messages[0].Body);
  await client.send(
    new DeleteMessageCommand({
      QueueUrl: queueUrl,
      ReceiptHandle: Messages[0].ReceiptHandle,
    }),
  );
} else {
  await client.send(
    new DeleteMessageBatchCommand({
      QueueUrl: queueUrl,
      Entries: Messages.map((message) => ({
        Id: message.MessageId,
        ReceiptHandle: message.ReceiptHandle,
      })),
    }),
  );
}
};
```

- Einzelheiten zur API finden Sie [DeleteMessageBatch](#) in der AWS SDK für JavaScript API-Referenz.

DeleteQueue

Das folgende Codebeispiel zeigt die Verwendung `DeleteQueue`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Löschen Sie eine Amazon-SQS-Warteschlange.


```
import { DeleteQueueCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "test-queue-url";

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const command = new DeleteQueueCommand({ QueueUrl: queueUrl });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [DeleteQueue](#) in der AWS SDK für JavaScript API-Referenz.

GetQueueAttributes

Das folgende Codebeispiel zeigt die Verwendung `GetQueueAttributes`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { GetQueueAttributesCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue-url";

export const getQueueAttributes = async (queueUrl = SQS_QUEUE_URL) => {
  const command = new GetQueueAttributesCommand({
    QueueUrl: queueUrl,
    AttributeNames: ["DelaySeconds"],
  });

  const response = await client.send(command);
```

```
console.log(response);
// {
//   '$metadata': {
//     httpStatusCode: 200,
//     requestId: '747a1192-c334-5682-a508-4cd5e8dc4e79',
//     extendedRequestId: undefined,
//     cfId: undefined,
//     attempts: 1,
//     totalRetryDelay: 0
//   },
//   Attributes: { DelaySeconds: '1' }
// }
return response;
};
```

- Einzelheiten zur API finden Sie [GetQueueAttributes](#) in der AWS SDK für JavaScript API-Referenz.

GetQueueUrl

Das folgende Codebeispiel zeigt die Verwendung `GetQueueUrl`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Rufen Sie die URL für eine Amazon-SQS-Warteschlange ab.

```
import { GetQueueUrlCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_NAME = "test-queue";

export const main = async (queueName = SQS_QUEUE_NAME) => {
  const command = new GetQueueUrlCommand({ QueueName: queueName });

  const response = await client.send(command);
```

```
console.log(response);
return response;
};
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [GetQueueUrl](#) in der AWS SDK für JavaScript API-Referenz.

ListQueues

Das folgende Codebeispiel zeigt die Verwendung `ListQueues`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Listen Sie Ihre Amazon-SQS-Warteschlangen auf.

```
import { paginateListQueues, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});

export const main = async () => {
  const paginatedListQueues = paginateListQueues({ client }, {});

  /** @type {string[]} */
  const urls = [];
  for await (const page of paginatedListQueues) {
    const nextUrls = page.QueueUrls?.filter((qurl) => !!qurl) || [];
    urls.push(...nextUrls);
    for (const url of urls) {
      console.log(url);
    }
  }

  return urls;
};
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [ListQueues](#) in der AWS SDK für JavaScript API-Referenz.

ReceiveMessage

Das folgende Codebeispiel zeigt die Verwendung `ReceiveMessage`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Empfangen Sie eine Nachricht von einer Amazon-SQS-Warteschlange.

```
import {
  ReceiveMessageCommand,
  DeleteMessageCommand,
  SQSClient,
  DeleteMessageBatchCommand,
} from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

const receiveMessage = (queueUrl) =>
  client.send(
    new ReceiveMessageCommand({
      AttributeNames: ["SentTimestamp"],
      MaxNumberOfMessages: 10,
      MessageAttributeNames: ["All"],
      QueueUrl: queueUrl,
      WaitTimeSeconds: 20,
      VisibilityTimeout: 20,
    }),
  );
```

```
export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const { Messages } = await receiveMessage(queueUrl);

  if (!Messages) {
    return;
  }

  if (Messages.length === 1) {
    console.log(Messages[0].Body);
    await client.send(
      new DeleteMessageCommand({
        QueueUrl: queueUrl,
        ReceiptHandle: Messages[0].ReceiptHandle,
      }),
    );
  } else {
    await client.send(
      new DeleteMessageBatchCommand({
        QueueUrl: queueUrl,
        Entries: Messages.map((message) => ({
          Id: message.MessageId,
          ReceiptHandle: message.ReceiptHandle,
        })),
      }),
    );
  }
};
```

Empfangen Sie mithilfe der Long-Poll-Unterstützung eine Nachricht aus einer Amazon-SQS-Warteschlange.

```
import { ReceiveMessageCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue-url";

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const command = new ReceiveMessageCommand({
    AttributeNames: ["SentTimestamp"],
    MaxNumberOfMessages: 1,
    MessageAttributeNames: ["All"],
    QueueUrl: queueUrl,
  });
```

```
// The duration (in seconds) for which the call waits for a message
// to arrive in the queue before returning. If a message is available,
// the call returns sooner than WaitTimeSeconds. If no messages are
// available and the wait time expires, the call returns successfully
// with an empty list of messages.
// https://docs.aws.amazon.com/AWSSimpleQueueService/latest/APIReference/
API_ReceiveMessage.html#API_ReceiveMessage_RequestSyntax
    WaitTimeSeconds: 20,
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Einzelheiten zur API finden Sie [ReceiveMessage](#) in der AWS SDK für JavaScript API-Referenz.

SendMessage

Das folgende Codebeispiel zeigt die Verwendung `SendMessage`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Senden Sie eine Nachricht an eine Amazon-SQS-Warteschlange.

```
import { SendMessageCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

export const main = async (sqsQueueUrl = SQS_QUEUE_URL) => {
  const command = new SendMessageCommand({
    QueueUrl: sqsQueueUrl,
    DelaySeconds: 10,
    MessageAttributes: {
```

```
Title: {
  DataType: "String",
  StringValue: "The Whistler",
},
Author: {
  DataType: "String",
  StringValue: "John Grisham",
},
WeeksOn: {
  DataType: "Number",
  StringValue: "6",
},
},
MessageBody:
  "Information about current NY Times fiction bestseller for week of
12/11/2016.",
});

const response = await client.send(command);
console.log(response);
return response;
};
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [SendMessage](#) in der AWS SDK für JavaScript API-Referenz.

SetQueueAttributes

Das folgende Codebeispiel zeigt die Verwendung `SetQueueAttributes`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { SetQueueAttributesCommand, SQSClient } from "@aws-sdk/client-sqs";
```

```
const client = new SQSClient({});
const SQS_QUEUE_URL = "queue-url";

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const command = new SetQueueAttributesCommand({
    QueueUrl: queueUrl,
    Attributes: {
      DelaySeconds: "1",
    },
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

Konfigurieren Sie eine Amazon-SQS-Warteschlange für die Verwendung einer Langabfrage.

```
import { SetQueueAttributesCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
const SQS_QUEUE_URL = "queue_url";

export const main = async (queueUrl = SQS_QUEUE_URL) => {
  const command = new SetQueueAttributesCommand({
    Attributes: {
      ReceiveMessageWaitTimeSeconds: "20",
    },
    QueueUrl: queueUrl,
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

Konfigurieren Sie eine Warteschlange für unzustellbare Nachrichten.

```
import { SetQueueAttributesCommand, SQSClient } from "@aws-sdk/client-sqs";

const client = new SQSClient({});
```



```
const SQS_QUEUE_URL = "queue_url";
const DEAD_LETTER_QUEUE_ARN = "dead_letter_queue_arn";

export const main = async (
  queueUrl = SQS_QUEUE_URL,
  deadLetterQueueArn = DEAD_LETTER_QUEUE_ARN,
) => {
  const command = new SetQueueAttributesCommand({
    Attributes: {
      RedrivePolicy: JSON.stringify({
        // Amazon SQS supports dead-letter queues (DLQ), which other
        // queues (source queues) can target for messages that can't
        // be processed (consumed) successfully.
        // https://docs.aws.amazon.com/AWSSimpleQueueService/latest/
        SQSDeveloperGuide/sqs-dead-letter-queues.html
        deadLetterTargetArn: deadLetterQueueArn,
        maxReceiveCount: "10",
      }),
    },
    QueueUrl: queueUrl,
  });

  const response = await client.send(command);
  console.log(response);
  return response;
};
```

- Einzelheiten zur API finden Sie [SetQueueAttributes](#) in der AWS SDK für JavaScript API-Referenz.

Szenarien

Erstellen Sie eine Amazon-Texttract-Explorer-Anwendung

Die folgenden Codebeispiele zeigen, wie die Amazon-Texttract-Ausgabe in einer interaktiven Anwendung untersucht werden kann.

SDK für JavaScript (v3)

Zeigt, wie Sie mit AWS SDK für JavaScript eine React-Anwendung erstellen, die Amazon Texttract verwendet, um Daten aus einem Dokumentbild zu extrahieren und auf einer interaktiven

Webseite anzuzeigen. Dieses Beispiel wird in einem Webbrowser ausgeführt und erfordert eine authentifizierte Amazon-Cognito-Identität für Anmeldeinformationen. Es verwendet Amazon Simple Storage Service (Amazon S3) zur Speicherung und fragt für Benachrichtigungen eine Amazon Simple Queue Service (Amazon SQS)-Warteschlange ab, die ein Amazon Simple Notification Service (Amazon SNS)-Thema abonniert hat.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

In diesem Beispiel verwendete Dienste

- Amazon Cognito Identity
- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon Textract

Veröffentlichen Sie Nachrichten in Warteschlangen

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Erstellen Sie ein Thema (FIFO oder Nicht-FIFO).
- Abonnieren Sie mehrere Warteschlangen für das Thema mit der Option, einen Filter anzuwenden.
- Veröffentlichen Sie eine Nachricht im Thema.
- Fragen Sie die Warteschlangen nach empfangenen Nachrichten ab.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Dies ist der Einstiegspunkt für dieses Szenario.

```
import { SNSClient } from "@aws-sdk/client-sns";
import { SQSClient } from "@aws-sdk/client-sqs";
```

```
import { TopicsQueuesWkflw } from "./TopicsQueuesWkflw.js";
import { Prompter } from "@aws-doc-sdk-examples/lib/prompter.js";

export const startSnsWorkflow = () => {
  const snsClient = new SNSClient({});
  const sqsClient = new SQSClient({});
  const prompter = new Prompter();
  const logger = console;

  const wkflw = new TopicsQueuesWkflw(snsClient, sqsClient, prompter, logger);

  wkflw.start();
};
```

Der obige Code stellt die erforderlichen Abhängigkeiten bereit und startet das Szenario. Der nächste Abschnitt enthält den Großteil des Beispiels.

```
const toneChoices = [
  { name: "cheerful", value: "cheerful" },
  { name: "funny", value: "funny" },
  { name: "serious", value: "serious" },
  { name: "sincere", value: "sincere" },
];

export class TopicsQueuesWkflw {
  // SNS topic is configured as First-In-First-Out
  isFifo = true;

  // Automatic content-based deduplication is enabled.
  autoDedup = false;

  snsClient;
  sqsClient;
  topicName;
  topicArn;
  subscriptionArns = [];
  /**
   * @type {{ queueName: string, queueArn: string, queueUrl: string, policy?:
   string }[]}
   */
}
```

```
 */
queues = [];
prompter;

/**
 * @param {import('@aws-sdk/client-sns').SNSClient} snsClient
 * @param {import('@aws-sdk/client-sqs').SQSClient} sqsClient
 * @param {import('../libs/prompter.js').Prompter} prompter
 * @param {import('../libs/logger.js').Logger} logger
 */
constructor(snsClient, sqsClient, prompter, logger) {
  this.snsClient = snsClient;
  this.sqsClient = sqsClient;
  this.prompter = prompter;
  this.logger = logger;
}

async welcome() {
  await this.logger.log(MESSAGES.description);
}

async confirmFifo() {
  await this.logger.log(MESSAGES.snsFifoDescription);
  this.isFifo = await this.prompter.confirm({
    message: MESSAGES.snsFifoPrompt,
  });

  if (this.isFifo) {
    this.logger.logSeparator(MESSAGES.headerDedup);
    await this.logger.log(MESSAGES.deduplicationNotice);
    await this.logger.log(MESSAGES.deduplicationDescription);
    this.autoDedup = await this.prompter.confirm({
      message: MESSAGES.deduplicationPrompt,
    });
  }
}

async createTopic() {
  await this.logger.log(MESSAGES.creatingTopics);
  this.topicName = await this.prompter.input({
    message: MESSAGES.topicNamePrompt,
  });
  if (this.isFifo) {
    this.topicName += ".fifo";
  }
}
```

```
    this.logger.logSeparator(MESSAGES.headerFifoNaming);
    await this.logger.log(MESSAGES.appendFifoNotice);
  }

  const response = await this.snsClient.send(
    new CreateTopicCommand({
      Name: this.topicName,
      Attributes: {
        FifoTopic: this.isFifo ? "true" : "false",
        ...(this.autoDedup ? { ContentBasedDeduplication: "true" } : {}),
      },
    }),
  );

  this.topicArn = response.TopicArn;

  await this.logger.log(
    MESSAGES.topicCreatedNotice
      .replace("${TOPIC_NAME}", this.topicName)
      .replace("${TOPIC_ARN}", this.topicArn),
  );
}

async createQueues() {
  await this.logger.log(MESSAGES.createQueuesNotice);
  // Increase this number to add more queues.
  const maxQueues = 2;

  for (let i = 0; i < maxQueues; i++) {
    await this.logger.log(MESSAGES.queueCount.replace("${COUNT}", i + 1));
    let queueName = await this.prompter.input({
      message: MESSAGES.queueNamePrompt.replace(
        "${EXAMPLE_NAME}",
        i === 0 ? "good-news" : "bad-news",
      ),
    });
  });

  if (this.isFifo) {
    queueName += ".fifo";
    await this.logger.log(MESSAGES.appendFifoNotice);
  }

  const response = await this.sqsClient.send(
    new CreateQueueCommand({
```

```

        QueueName: queueName,
        Attributes: { ...(this.isFifo ? { FifoQueue: "true" } : {}) },
    })),
    );

    const { Attributes } = await this.sqsClient.send(
        new GetQueueAttributesCommand({
            QueueUrl: response.QueueUrl,
            AttributeNames: ["QueueArn"],
        })),
    );

    this.queues.push({
        queueName,
        queueArn: Attributes.QueueArn,
        queueUrl: response.QueueUrl,
    });

    await this.logger.log(
        MESSAGES.queueCreatedNotice
            .replace("${QUEUE_NAME}", queueName)
            .replace("${QUEUE_URL}", response.QueueUrl)
            .replace("${QUEUE_ARN}", Attributes.QueueArn),
    );
}
}

async attachQueueIamPolicies() {
    for (const [index, queue] of this.queues.entries()) {
        const policy = JSON.stringify(
            {
                Statement: [
                    {
                        Effect: "Allow",
                        Principal: {
                            Service: "sns.amazonaws.com",
                        },
                        Action: "sqs:SendMessage",
                        Resource: queue.queueArn,
                        Condition: {
                            ArnEquals: {
                                "aws:SourceArn": this.topicArn,
                            },
                        },
                    },
                ],
            },
        );
    }
}

```

```
        },
      ],
    },
    null,
    2,
  );

  if (index !== 0) {
    this.logger.logSeparator();
  }

  await this.logger.log(MESSAGES.attachPolicyNotice);
  console.log(policy);
  const addPolicy = await this.prompter.confirm({
    message: MESSAGES.addPolicyConfirmation.replace(
      "${QUEUE_NAME}",
      queue.queueName,
    ),
  });

  if (addPolicy) {
    await this.sqsClient.send(
      new SetQueueAttributesCommand({
        QueueUrl: queue.queueUrl,
        Attributes: {
          Policy: policy,
        },
      }),
    );
    queue.policy = policy;
  } else {
    await this.logger.log(
      MESSAGES.policyNotAttachedNotice.replace(
        "${QUEUE_NAME}",
        queue.queueName,
      ),
    );
  }
}

async subscribeQueuesToTopic() {
  for (const [index, queue] of this.queues.entries()) {
    /**
```

```

    * @type {import('@aws-sdk/client-sns').SubscribeCommandInput}
    */
    const subscribeParams = {
      TopicArn: this.topicArn,
      Protocol: "sqs",
      Endpoint: queue.queueArn,
    };
    let tones = [];

    if (this.isFifo) {
      if (index === 0) {
        await this.logger.log(MESSAGES.fifoFilterNotice);
      }
      tones = await this.prompter.checkbox({
        message: MESSAGES.fifoFilterSelect.replace(
          "${QUEUE_NAME}",
          queue.queueName,
        ),
        choices: toneChoices,
      });

      if (tones.length) {
        subscribeParams.Attributes = {
          FilterPolicyScope: "MessageAttributes",
          FilterPolicy: JSON.stringify({
            tone: tones,
          }),
        };
      }
    }

    const { SubscriptionArn } = await this.snsClient.send(
      new SubscribeCommand(subscribeParams),
    );

    this.subscriptionArns.push(SubscriptionArn);

    await this.logger.log(
      MESSAGES.queueSubscribedNotice
        .replace("${QUEUE_NAME}", queue.queueName)
        .replace("${TOPIC_NAME}", this.topicName)
        .replace("${TONES}", tones.length ? tones.join(", ") : "none"),
    );
  }

```



```
}

async publishMessages() {
  const message = await this.prompter.input({
    message: MESSAGES.publishMessagePrompt,
  });

  let groupId;
  let deduplicationId;
  let choices;

  if (this.isFifo) {
    await this.logger.log(MESSAGES.groupIdNotice);
    groupId = await this.prompter.input({
      message: MESSAGES.groupIdPrompt,
    });
  }

  if (this.autoDedup === false) {
    await this.logger.log(MESSAGES.deduplicationIdNotice);
    deduplicationId = await this.prompter.input({
      message: MESSAGES.deduplicationIdPrompt,
    });
  }

  choices = await this.prompter.checkbox({
    message: MESSAGES.messageAttributesPrompt,
    choices: toneChoices,
  });
}

await this.snsClient.send(
  new PublishCommand({
    TopicArn: this.topicArn,
    Message: message,
    ...(groupId
      ? {
          MessageGroupId: groupId,
        }
      : {}),
    ...(deduplicationId
      ? {
          MessageDeduplicationId: deduplicationId,
        }
      : {}),
  })
);
```

```
        ...(choices
        ? {
            MessageAttributes: {
                tone: {
                    DataType: "String.Array",
                    StringValue: JSON.stringify(choices),
                },
            },
        }
        : {})),
    )),
);

const publishAnother = await this.prompter.confirm({
    message: MESSAGES.publishAnother,
});

if (publishAnother) {
    await this.publishMessages();
}

async receiveAndDeleteMessages() {
    for (const queue of this.queues) {
        const { Messages } = await this.sqsClient.send(
            new ReceiveMessageCommand({
                QueueUrl: queue.queueUrl,
            }),
        );

        if (Messages) {
            await this.logger.log(
                MESSAGES.messagesReceivedNotice.replace(
                    "${QUEUE_NAME}",
                    queue.queueName,
                ),
            );

            console.log(Messages);

            await this.sqsClient.send(
                new DeleteMessageBatchCommand({
                    QueueUrl: queue.queueUrl,
                    Entries: Messages.map((message) => ({
                        Id: message.MessageId,
```

```
        ReceiptHandle: message.ReceiptHandle,
      })),
    }),
  );
} else {
  await this.logger.log(
    MESSAGES.noMessagesReceivedNotice.replace(
      "${QUEUE_NAME}",
      queue.queueName,
    ),
  );
}
}

const deleteAndPoll = await this.prompter.confirm({
  message: MESSAGES.deleteAndPollConfirmation,
});

if (deleteAndPoll) {
  await this.receiveAndDeleteMessages();
}
}

async destroyResources() {
  for (const subscriptionArn of this.subscriptionArns) {
    await this.snsClient.send(
      new UnsubscribeCommand({ SubscriptionArn: subscriptionArn }),
    );
  }

  for (const queue of this.queues) {
    await this.sqsClient.send(
      new DeleteQueueCommand({ QueueUrl: queue.queueUrl }),
    );
  }

  if (this.topicArn) {
    await this.snsClient.send(
      new DeleteTopicCommand({ TopicArn: this.topicArn }),
    );
  }
}

async start() {
```

```
console.clear();

try {
  this.logger.logSeparator(MESSAGES.headerWelcome);
  await this.welcome();
  this.logger.logSeparator(MESSAGES.headerFifo);
  await this.confirmFifo();
  this.logger.logSeparator(MESSAGES.headerCreateTopic);
  await this.createTopic();
  this.logger.logSeparator(MESSAGES.headerCreateQueues);
  await this.createQueues();
  this.logger.logSeparator(MESSAGES.headerAttachPolicy);
  await this.attachQueueIamPolicies();
  this.logger.logSeparator(MESSAGES.headerSubscribeQueues);
  await this.subscribeQueuesToTopic();
  this.logger.logSeparator(MESSAGES.headerPublishMessage);
  await this.publishMessages();
  this.logger.logSeparator(MESSAGES.headerReceiveMessages);
  await this.receiveAndDeleteMessages();
} catch (err) {
  console.error(err);
} finally {
  await this.destroyResources();
}
}
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für JavaScript - API-Referenz.
 - [CreateQueue](#)
 - [CreateTopic](#)
 - [DeleteMessageBatch](#)
 - [DeleteQueue](#)
 - [DeleteTopic](#)
 - [GetQueueAttributes](#)
 - [Veröffentlichen](#)
 - [ReceiveMessage](#)
 - [SetQueueAttributes](#)

- [Abonnieren](#)
- [Unsubscribe](#)

Serverless-Beispiele

Aufrufen einer Lambda-Funktion über einen Amazon-SQS-Auslöser

Das folgende Codebeispiel zeigt, wie eine Lambda-Funktion implementiert wird, die ein Ereignis empfängt, das durch den Empfang von Nachrichten aus einer SQS-Warteschlange ausgelöst wird. Die Funktion ruft die Nachrichten aus dem Ereignisparameter ab und protokolliert den Inhalt jeder Nachricht.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Konsumieren eines SQS-Ereignisses mit Lambda unter Verwendung. JavaScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
exports.handler = async (event, context) => {
  for (const message of event.Records) {
    await processMessageAsync(message);
  }
  console.info("done");
};

async function processMessageAsync(message) {
  try {
    console.log(`Processed message ${message.body}`);
    // TODO: Do interesting work based on the new message
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

```
}
```

Konsumieren eines SQS-Ereignisses mit Lambda unter Verwendung. TypeScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { SQSEvent, Context, SQSHandler, SQSRecord } from "aws-lambda";

export const functionHandler: SQSHandler = async (
  event: SQSEvent,
  context: Context
): Promise<void> => {
  for (const message of event.Records) {
    await processMessageAsync(message);
  }
  console.info("done");
};

async function processMessageAsync(message: SQSRecord): Promise<any> {
  try {
    console.log(`Processed message ${message.body}`);
    // TODO: Do interesting work based on the new message
    await Promise.resolve(1); //Placeholder for actual async work
  } catch (err) {
    console.error("An error occurred");
    throw err;
  }
}
```

Melden von Batch-Elementfehlern für Lambda-Funktionen mit einem Amazon-SQS-Auslöser

Das folgende Codebeispiel zeigt, wie eine teilweise Batch-Antwort für Lambda-Funktionen implementiert wird, die Ereignisse aus einer SQS-Warteschlange empfangen. Die Funktion meldet die Batch-Elementfehler in der Antwort und signalisiert Lambda, diese Nachrichten später erneut zu versuchen.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit [Serverless-Beispielen](#).

Melden von SQS-Batch-Elementfehlern mit Lambda unter Verwendung von JavaScript

```
// Node.js 20.x Lambda runtime, AWS SDK for Javascript V3
export const handler = async (event, context) => {
  const batchItemFailures = [];
  for (const record of event.Records) {
    try {
      await processMessageAsync(record, context);
    } catch (error) {
      batchItemFailures.push({ itemIdentifier: record.messageId });
    }
  }
  return { batchItemFailures };
};

async function processMessageAsync(record, context) {
  if (record.body && record.body.includes("error")) {
    throw new Error("There is an error in the SQS Message.");
  }
  console.log(`Processed message: ${record.body}`);
}
```

Melden von SQS-Batch-Elementfehlern mit Lambda unter Verwendung von TypeScript

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import { SQSEvent, SQSBatchResponse, Context, SQSBatchItemFailure, SQSRecord } from
  'aws-lambda';

export const handler = async (event: SQSEvent, context: Context):
  Promise<SQSBatchResponse> => {
  const batchItemFailures: SQSBatchItemFailure[] = [];

  for (const record of event.Records) {
```

```
        try {
            await processMessageAsync(record);
        } catch (error) {
            batchItemFailures.push({ itemIdentifier: record.messageId });
        }
    }

    return {batchItemFailures: batchItemFailures};
};

async function processMessageAsync(record: SQSRecord): Promise<void> {
    if (record.body && record.body.includes("error")) {
        throw new Error('There is an error in the SQS Message.');
```

Step Functions Functions-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie die Funktionen AWS SDK für JavaScript (v3) mit Step verwenden.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kodex finden.

Themen

- [Aktionen](#)

Aktionen

StartExecution

Das folgende Codebeispiel zeigt die Verwendung von `StartExecution`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { SFNClient, StartExecutionCommand } from "@aws-sdk/client-sfn";

/**
 * @param {{ sfnClient: SFNClient, stateMachineArn: string }} config
 */
export async function startExecution({ sfnClient, stateMachineArn }) {
  const response = await sfnClient.send(
    new StartExecutionCommand({
      stateMachineArn,
    }),
  );
  console.log(response);
  // Example response:
  // {
  //   '$metadata': {
  //     httpStatusCode: 200,
  //     requestId: '202a9309-c16a-454b-adeb-c4d19afe3bf2',
  //     extendedRequestId: undefined,
  //     cfId: undefined,
  //     attempts: 1,
  //     totalRetryDelay: 0
  //   },
  //   executionArn: 'arn:aws:states:us-east-1:000000000000:execution:MyStateMachine:aaaaaaaa-f787-49fb-a20c-1b61c64eafe6',
  //   startDate: 2024-01-04T15:54:08.362Z
  // }
  return response;
}

// Call function if run directly
import { fileURLToPath } from "node:url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  startExecution({ sfnClient: new SFNClient({}), stateMachineArn: "ARN" });
}
```

```
}
```

- Einzelheiten zur API finden Sie [StartExecution](#) in der AWS SDK für JavaScript API-Referenz.

AWS STS Beispiele für die Verwendung von SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie AWS SDK für JavaScript (v3) mit verwenden AWS STS.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kodex finden.

Themen

- [Aktionen](#)

Aktionen

AssumeRole

Das folgende Codebeispiel zeigt die Verwendung von `AssumeRole`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie den Client.

```
import { STSClient } from "@aws-sdk/client-sts";
```

```
// Set the AWS Region.
const REGION = "us-east-1";
// Create an AWS STS service client object.
export const client = new STSClient({ region: REGION });
```

Übernehmen Sie die IAM-Rolle.

```
import { AssumeRoleCommand } from "@aws-sdk/client-sts";

import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Returns a set of temporary security credentials that you can use to
    // access Amazon Web Services resources that you might not normally
    // have access to.
    const command = new AssumeRoleCommand({
      // The Amazon Resource Name (ARN) of the role to assume.
      RoleArn: "ROLE_ARN",
      // An identifier for the assumed role session.
      RoleSessionName: "session1",
      // The duration, in seconds, of the role session. The value specified
      // can range from 900 seconds (15 minutes) up to the maximum session
      // duration set for the role.
      DurationSeconds: 900,
    });
    const response = await client.send(command);
    console.log(response);
  } catch (err) {
    console.error(err);
  }
};
```

- Einzelheiten zur API finden Sie [AssumeRole](#) in der AWS SDK für JavaScript API-Referenz.

Support Beispiele für die Verwendung von SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie AWS SDK für JavaScript (v3) mit verwenden Support.

Bei Grundlagen handelt es sich um Codebeispiele, die Ihnen zeigen, wie Sie die wesentlichen Vorgänge innerhalb eines Services ausführen.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kodex finden.

Themen

- [Erste Schritte](#)
- [Grundlagen](#)
- [Aktionen](#)

Erste Schritte

Hallo Support

Das folgende Codebeispiel veranschaulicht, wie Sie mit der Verwendung von Support beginnen.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Rufen Sie „main()“ auf, um das Beispiel auszuführen.

```
import {
  DescribeServicesCommand,
  SupportClient,
} from "@aws-sdk/client-support";

// Change the value of 'region' to your preferred AWS Region.
const client = new SupportClient({ region: "us-east-1" });
```

```
const getServiceCount = async () => {
  try {
    const { services } = await client.send(new DescribeServicesCommand({}));
    return services.length;
  } catch (err) {
    if (err.name === "SubscriptionRequiredException") {
      throw new Error(
        "You must be subscribed to the AWS Support plan to use this feature.",
      );
    }
    throw err;
  }
};

export const main = async () => {
  try {
    const count = await getServiceCount();
    console.log(`Hello, AWS Support! There are ${count} services available.`);
  } catch (err) {
    console.error("Failed to get service count: ", err.message);
  }
};
```

- Einzelheiten zur API finden Sie [DescribeServices](#) in der AWS SDK für JavaScript API-Referenz.

Grundlagen

Kennenlernen der Grundlagen

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Rufen Sie verfügbare Services und Schweregrade für Fälle ab und zeigen Sie sie an.
- Erstellen Sie einen Supportfall mit einem ausgewählten Service, einer ausgewählten Kategorie und einem ausgewählten Schweregrad.
- Rufen Sie eine Liste der offenen Fälle für den aktuellen Tag ab und zeigen Sie sie an.
- Fügen Sie dem neuen Fall einen Anhangssatz und eine Mitteilung hinzu.
- Beschreiben Sie den neuen Anhang und die Mitteilung für den Fall.
- Lösen Sie den Fall.
- Rufen Sie eine Liste der gelösten Fälle für den aktuellen Tag ab und zeigen Sie sie an.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Führen Sie ein interaktives Szenario im Terminal aus.

```
import {
  AddAttachmentsToSetCommand,
  AddCommunicationToCaseCommand,
  CreateCaseCommand,
  DescribeAttachmentCommand,
  DescribeCasesCommand,
  DescribeCommunicationsCommand,
  DescribeServicesCommand,
  DescribeSeverityLevelsCommand,
  ResolveCaseCommand,
  SupportClient,
} from "@aws-sdk/client-support";
import * as inquirer from "@inquirer/prompts";
import { retry } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";

const wrapText = (text, char = "=") => {
  const rule = char.repeat(80);
  return `${rule}\n    ${text}\n${rule}\n`;
};

const client = new SupportClient({ region: "us-east-1" });

// Verify that the account has a Support plan.
export const verifyAccount = async () => {
  const command = new DescribeServicesCommand({});

  try {
    await client.send(command);
  } catch (err) {
    if (err.name === "SubscriptionRequiredException") {
      throw new Error(
        "You must be subscribed to the AWS Support plan to use this feature.",
      );
    }
  }
}
```

```

    }
    throw err;
  }
};

/**
 * Select a service from the list returned from DescribeServices.
 */
export const getService = async () => {
  const { services } = await client.send(new DescribeServicesCommand({}));
  const selectedService = await inquirer.select({
    message:
      "Select a service. Your support case will be created for this service. The
      list of services is truncated for readability.",
    choices: services.slice(0, 10).map((s) => ({ name: s.name, value: s })),
  });
  return selectedService;
};

/**
 * @param {{ categories: import('@aws-sdk/client-support').Category[]}} service
 */
export const getCategory = async (service) => {
  const selectedCategory = await inquirer.select({
    message: "Select a category.",
    choices: service.categories.map((c) => ({ name: c.name, value: c })),
  });
  return selectedCategory;
};

// Get the available severity levels for the account.
export const getSeverityLevel = async () => {
  const command = new DescribeSeverityLevelsCommand({});
  const { severityLevels } = await client.send(command);
  const selectedSeverityLevel = await inquirer.select({
    message: "Select a severity level.",
    choices: severityLevels.map((s) => ({ name: s.name, value: s })),
  });
  return selectedSeverityLevel;
};

/**
 * Create a new support case
 * @param {{

```

```
* selectedService: import('@aws-sdk/client-support').Service
* selectedCategory: import('@aws-sdk/client-support').Category
* selectedSeverityLevel: import('@aws-sdk/client-support').SeverityLevel
* }} selections
* @returns
*/
export const createCase = async ({
  selectedService,
  selectedCategory,
  selectedSeverityLevel,
}) => {
  const command = new CreateCaseCommand({
    subject: "IGNORE: Test case",
    communicationBody: "This is a test. Please ignore.",
    serviceCode: selectedService.code,
    categoryCode: selectedCategory.code,
    severityCode: selectedSeverityLevel.code,
  });
  const { caseId } = await client.send(command);
  return caseId;
};

// Get a list of open support cases created today.
export const getTodayOpenCases = async () => {
  const d = new Date();
  const startOfToday = new Date(d.getFullYear(), d.getMonth(), d.getDate());
  const command = new DescribeCasesCommand({
    includeCommunications: false,
    afterTime: startOfToday.toISOString(),
  });

  const { cases } = await client.send(command);

  if (cases.length === 0) {
    throw new Error(
      "Unexpected number of cases. Expected more than 0 open cases.",
    );
  }
  return cases;
};

// Create an attachment set.
export const createAttachmentSet = async () => {
  const command = new AddAttachmentsToSetCommand({
```



```
    attachments: [
      {
        fileName: "example.txt",
        data: new TextEncoder().encode("some example text"),
      },
    ],
  });
  const { attachmentSetId } = await client.send(command);
  return attachmentSetId;
};

export const linkAttachmentSetToCase = async (attachmentSetId, caseId) => {
  const command = new AddCommunicationToCaseCommand({
    attachmentSetId,
    caseId,
    communicationBody: "Adding attachment set to case.",
  });
  await client.send(command);
};

// Get all communications for a support case.
export const getCommunications = async (caseId) => {
  const command = new DescribeCommunicationsCommand({
    caseId,
  });
  const { communications } = await client.send(command);
  return communications;
};

/**
 * @param {import('@aws-sdk/client-support').Communication[]} communications
 */
export const getFirstAttachment = (communications) => {
  const firstCommWithAttachment = communications.find(
    (c) => c.attachmentSet.length > 0,
  );
  return firstCommWithAttachment?.attachmentSet[0].attachmentId;
};

// Get an attachment.
export const getAttachment = async (attachmentId) => {
  const command = new DescribeAttachmentCommand({
    attachmentId,
  });
};
```

```

    const { attachment } = await client.send(command);
    return attachment;
  };

  // Resolve the case matching the given case ID.
  export const resolveCase = async (caseId) => {
    const shouldResolve = await inquirer.confirm({
      message: `Do you want to resolve ${caseId}?`,
    });

    if (shouldResolve) {
      const command = new ResolveCaseCommand({
        caseId: caseId,
      });

      await client.send(command);
      return true;
    }
    return false;
  };

  /**
   * Find a specific case in the list of provided cases by case ID.
   * If the case is not found, and the results are paginated, continue
   * paging through the results.
   * @param {{
   *   caseId: string,
   *   cases: import('@aws-sdk/client-support').CaseDetails[]
   *   nextToken: string
   * }} options
   * @returns
   */
  export const findCase = async ({ caseId, cases, nextToken }) => {
    const foundCase = cases.find((c) => c.caseId === caseId);

    if (foundCase) {
      return foundCase;
    }

    if (nextToken) {
      const response = await client.send(
        new DescribeCasesCommand({
          nextToken,
          includeResolvedCases: true,

```

```
    }},
  );
  return findCase({
    caseId,
    cases: response.cases,
    nextToken: response.nextToken,
  });
}

throw new Error(`${caseId} not found.`);
};

// Get all cases created today.
export const getTodaysResolvedCases = async (caseIdToWaitFor) => {
  const d = new Date("2023-01-18");
  const startOfToday = new Date(d.getFullYear(), d.getMonth(), d.getDate());
  const command = new DescribeCasesCommand({
    includeCommunications: false,
    afterTime: startOfToday.toISOString(),
    includeResolvedCases: true,
  });
  const { cases, nextToken } = await client.send(command);
  await findCase({ cases, caseId: caseIdToWaitFor, nextToken });
  return cases.filter((c) => c.status === "resolved");
};

const main = async () => {
  let caseId;
  try {
    console.log(wrapText("Welcome to the AWS Support basic usage scenario."));

    // Verify that the account is subscribed to support.
    await verifyAccount();

    // Provided a truncated list of services and prompt the user to select one.
    const selectedService = await getService();

    // Provided the categories for the selected service and prompt the user to
    select one.
    const selectedCategory = await getCategory(selectedService);

    // Provide the severity available severity levels for the account and prompt the
    user to select one.
    const selectedSeverityLevel = await getSeverityLevel();
```

```
// Create a support case.
console.log("\nCreating a support case.");
caseId = await createCase({
  selectedService,
  selectedCategory,
  selectedSeverityLevel,
});
console.log(`Support case created: ${caseId}`);

// Display a list of open support cases created today.
const todaysOpenCases = await retry(
  { intervalInMs: 1000, maxRetries: 15 },
  getTodaysOpenCases,
);
console.log(
  `\nOpen support cases created today: ${todaysOpenCases.length}`,
);
console.log(todaysOpenCases.map((c) => `${c.caseId}`).join("\n"));

// Create an attachment set.
console.log("\nCreating an attachment set.");
const attachmentSetId = await createAttachmentSet();
console.log(`Attachment set created: ${attachmentSetId}`);

// Add the attachment set to the support case.
console.log(`\nAdding attachment set to ${caseId}`);
await linkAttachmentSetToCase(attachmentSetId, caseId);
console.log(`Attachment set added to ${caseId}`);

// List the communications for a support case.
console.log(`\nListing communications for ${caseId}`);
const communications = await getCommunications(caseId);
console.log(
  communications
    .map(
      (c) =>
        `Communication created on ${c.timeCreated}. Has
        ${c.attachmentSet.length} attachments.`,
    )
    .join("\n"),
);

// Describe the first attachment.
```

```
console.log(`\nDescribing attachment ${attachmentSetId}`);
const attachmentId = getFirstAttachment(communications);
const attachment = await getAttachment(attachmentId);
console.log(
  `Attachment is the file '${
    attachment.fileName
  }' with data: \n${new TextDecoder().decode(attachment.data)}`,
);

// Confirm that the support case should be resolved.
const isResolved = await resolveCase(caseId);
if (isResolved) {
  // List the resolved cases and include the one previously created.
  // Resolved cases can take a while to appear.
  console.log(
    "\nWaiting for case status to be marked as resolved. This can take some
time.",
  );
  const resolvedCases = await retry(
    { intervalInMs: 20000, maxRetries: 15 },
    () => getTodayResolvedCases(caseId),
  );
  console.log("Resolved cases:");
  console.log(resolvedCases.map((c) => c.caseId).join("\n"));
}
} catch (err) {
  console.error(err);
}
};
```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für JavaScript - API-Referenz.
 - [AddAttachmentsToSet](#)
 - [AddCommunicationToCase](#)
 - [CreateCase](#)
 - [DescribeAttachment](#)
 - [DescribeCases](#)
 - [DescribeCommunications](#)
 - [DescribeServices](#)

- [DescribeSeverityLevels](#)
- [ResolveCase](#)

Aktionen

AddAttachmentsToSet

Das folgende Codebeispiel zeigt die Verwendung `AddAttachmentsToSet`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { AddAttachmentsToSetCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Create a new attachment set or add attachments to an existing set.
    // Provide an 'attachmentSetId' value to add attachments to an existing set.
    // Use AddCommunicationToCase or CreateCase to associate an attachment set with
    a support case.
    const response = await client.send(
      new AddAttachmentsToSetCommand({
        // You can add up to three attachments per set. The size limit is 5 MB per
        attachment.
        attachments: [
          {
            fileName: "example.txt",
            data: new TextEncoder().encode("some example text"),
          },
        ],
      }),
    );
    // Use this ID in AddCommunicationToCase or CreateCase.
    console.log(response.attachmentSetId);
  }
}
```

```
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

- Einzelheiten zur API finden Sie [AddAttachmentsToSet](#) in der AWS SDK für JavaScript API-Referenz.

AddCommunicationToCase

Das folgende Codebeispiel zeigt die Verwendung `AddCommunicationToCase`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { AddCommunicationToCaseCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";

export const main = async () => {
  let attachmentSetId;

  try {
    // Add a communication to a case.
    const response = await client.send(
      new AddCommunicationToCaseCommand({
        communicationBody: "Adding an attachment.",
        // Set value to an existing support case id.
        caseId: "CASE_ID",
        // Optional. Set value to an existing attachment set id to add attachments
        // to the case.
        attachmentSetId,
      })
    );
  }
```

```
    console.log(response);  
    return response;  
  } catch (err) {  
    console.error(err);  
  }  
};
```

- Einzelheiten zur API finden Sie [AddCommunicationToCase](#) in der AWS SDK für JavaScript API-Referenz.

CreateCase

Das folgende Codebeispiel zeigt die Verwendung `CreateCase`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { CreateCaseCommand } from "@aws-sdk/client-support";  
  
import { client } from "../libs/client.js";  
  
export const main = async () => {  
  try {  
    // Create a new case and log the case id.  
    // Important: This creates a real support case in your account.  
    const response = await client.send(  
      new CreateCaseCommand({  
        // The subject line of the case.  
        subject: "IGNORE: Test case",  
        // Use DescribeServices to find available service codes for each service.  
        serviceCode: "service-quicksight-end-user",  
        // Use DescribeSecurityLevels to find available severity codes for your  
        support plan.  
        severityCode: "low",  
        // Use DescribeServices to find available category codes for each service.  

```



```

        categoryCode: "end-user-support",
        // The main description of the support case.
        communicationBody: "This is a test. Please ignore.",
    }},
    );
    console.log(response.caseId);
    return response;
} catch (err) {
    console.error(err);
}
};

```

- Einzelheiten zur API finden Sie [CreateCase](#) in der AWS SDK für JavaScript API-Referenz.

DescribeAttachment

Das folgende Codebeispiel zeigt die Verwendung `DescribeAttachment`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```

import { DescribeAttachmentCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";

export const main = async () => {
    try {
        // Get the metadata and content of an attachment.
        const response = await client.send(
            new DescribeAttachmentCommand({
                // Set value to an existing attachment id.
                // Use DescribeCommunications or DescribeCases to find an attachment id.
                attachmentId: "ATTACHMENT_ID",
            }),
        );
    }
};

```

```
    console.log(response.attachment?.fileName);
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

- Einzelheiten zur API finden Sie [DescribeAttachment](#) in der AWS SDK für JavaScript API-Referenz.

DescribeCases

Das folgende Codebeispiel zeigt die Verwendung `DescribeCases`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { DescribeCasesCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";

export const main = async () => {
  try {
    // Get all of the unresolved cases in your account.
    // Filter or expand results by providing parameters to the DescribeCasesCommand.
    Refer
    // to the TypeScript definition and the API doc for more information on possible
    parameters.
    // https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/clients/client-
    support/interfaces/describecasescommandinput.html
    const response = await client.send(new DescribeCasesCommand({}));
    const caseIds = response.cases.map((supportCase) => supportCase.caseId);
    console.log(caseIds);
    return response;
  } catch (err) {
```

```
    console.error(err);  
  }  
};
```

- Einzelheiten zur API finden Sie [DescribeCases](#) in der AWS SDK für JavaScript API-Referenz.

DescribeCommunications

Das folgende Codebeispiel zeigt die Verwendung `DescribeCommunications`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { DescribeCommunicationsCommand } from "@aws-sdk/client-support";  
  
import { client } from "../libs/client.js";  
  
export const main = async () => {  
  try {  
    // Get all communications for the support case.  
    // Filter results by providing parameters to the DescribeCommunicationsCommand.  
    Refer  
    // to the TypeScript definition and the API doc for more information on possible  
    parameters.  
    // https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/clients/client-  
    support/interfaces/describecommunicationscommandinput.html  
    const response = await client.send(  
      new DescribeCommunicationsCommand({  
        // Set value to an existing case id.  
        caseId: "CASE_ID",  
      }),  
    );  
    const text = response.communications.map((item) => item.body).join("\n");  
    console.log(text);  
    return response;  
  }  
};
```

```
    } catch (err) {  
      console.error(err);  
    }  
  };  
};
```

- Einzelheiten zur API finden Sie [DescribeCommunications](#) in der AWS SDK für JavaScript API-Referenz.

DescribeSeverityLevels

Das folgende Codebeispiel zeigt die Verwendung `DescribeSeverityLevels`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { DescribeSeverityLevelsCommand } from "@aws-sdk/client-support";  
  
import { client } from "../libs/client.js";  
  
export const main = async () => {  
  try {  
    // Get the list of severity levels.  
    // The available values depend on the support plan for the account.  
    const response = await client.send(new DescribeSeverityLevelsCommand({}));  
    console.log(response.severityLevels);  
    return response;  
  } catch (err) {  
    console.error(err);  
  }  
};
```

- Einzelheiten zur API finden Sie [DescribeSeverityLevels](#) in der AWS SDK für JavaScript API-Referenz.

ResolveCase

Das folgende Codebeispiel zeigt die Verwendung `ResolveCase`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { ResolveCaseCommand } from "@aws-sdk/client-support";

import { client } from "../libs/client.js";

const main = async () => {
  try {
    const response = await client.send(
      new ResolveCaseCommand({
        caseId: "CASE_ID",
      }),
    );

    console.log(response.finalCaseStatus);
    return response;
  } catch (err) {
    console.error(err);
  }
};
```

- Einzelheiten zur API finden Sie [ResolveCase](#) in der AWS SDK für JavaScript API-Referenz.

Systems Manager Manager-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK für JavaScript (v3) mit Systems Manager Aktionen ausführen und allgemeine Szenarien implementieren.

Bei Grundlagen handelt es sich um Codebeispiele, die Ihnen zeigen, wie Sie die wesentlichen Vorgänge innerhalb eines Services ausführen.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kodex finden.

Themen

- [Erste Schritte](#)
- [Grundlagen](#)
- [Aktionen](#)

Erste Schritte

Hallo Systems Manager

Das folgende Codebeispiel zeigt die ersten Schritte mit Systems Manager.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { paginateListDocuments, SSMClient } from "@aws-sdk/client-ssm";

// Call ListDocuments and display the result.
export const main = async () => {
  const client = new SSMClient();
  const listDocumentsPaginated = [];
  console.log(
    "Hello, AWS Systems Manager! Let's list some of your documents:\n",
  );
  try {
    // The paginate function is a wrapper around the base command.
    const paginator = paginateListDocuments({ client }, { MaxResults: 5 });
    for await (const page of paginator) {
```

```
        listDocumentsPaginated.push(...page.DocumentIdentifiers);
    }
} catch (caught) {
    console.error(`There was a problem saying hello: ${caught.message}`);
    throw caught;
}

for (const { Name, DocumentFormat, CreatedDate } of listDocumentsPaginated) {
    console.log(`${Name} - ${DocumentFormat} - ${CreatedDate}`);
}
};

// Call function if run directly.
import { fileURLToPath } from "node:url";
if (process.argv[1] === fileURLToPath(import.meta.url)) {
    main();
}
```

- Einzelheiten zur API finden Sie [ListDocuments](#) in der AWS SDK für JavaScript API-Referenz.

Grundlagen

Kennenlernen der Grundlagen

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Erstellen Sie ein Wartungsfenster.
- Ändern Sie den Zeitplan des Wartungsfensters.
- Erstellen Sie ein Dokument.
- Sendet einen Befehl an eine angegebene EC2 Instanz.
- Erstellen Sie eine OpsItem.
- Aktualisieren und lösen Sie das OpsItem.
- Löschen Sie das Wartungsfenster OpsItem, und das Dokument.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import {
  Scenario,
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/index.js";
import { fileURLToPath } from "node:url";
import {
  CreateDocumentCommand,
  CreateMaintenanceWindowCommand,
  CreateOpsItemCommand,
  DeleteDocumentCommand,
  DeleteMaintenanceWindowCommand,
  DeleteOpsItemCommand,
  DescribeOpsItemsCommand,
  DocumentAlreadyExists,
  OpsItemStatus,
  waitUntilCommandExecuted,
  CancelCommandCommand,
  paginateListCommandInvocations,
  SendCommandCommand,
  UpdateMaintenanceWindowCommand,
  UpdateOpsItemCommand,
  SSMClient,
} from "@aws-sdk/client-ssm";
import { parseArgs } from "node:util";

/**
 * @typedef {{
 *   ssmClient: import('@aws-sdk/client-ssm').SSMClient,
 *   documentName?: string
 *   maintenanceWindow?: string
 *   winId?: int
 *   ec2InstanceId?: string
```



```
*   requestedDateTime?: Date
*   opsItemId?: string
*   askToDeleteResources?: boolean
* }} State
*/

const defaultMaintenanceWindow = "ssm-maintenance-window";
const defaultDocumentName = "ssmdocument";
// The timeout duration is highly dependent on the specific setup and environment
// necessary. This example handles only the most common error cases, and uses a much
// shorter duration than most productions systems would use.
const COMMAND_TIMEOUT_DURATION_SECONDS = 30; // 30 seconds

const pressEnter = new ScenarioInput("continue", "Press Enter to continue", {
  type: "confirm",
});

const greet = new ScenarioOutput(
  "greet",
  `Welcome to the AWS Systems Manager SDK Getting Started scenario.
  This program demonstrates how to interact with Systems Manager using the AWS SDK
  for JavaScript V3.
  Systems Manager is the operations hub for your AWS applications and resources
  and a secure end-to-end management solution.
  The program's primary functions include creating a maintenance window, creating
  a document, sending a command to a document,
  listing documents, listing commands, creating an OpsItem, modifying an OpsItem,
  and deleting Systems Manager resources.
  Upon completion of the program, all AWS resources are cleaned up.
  Let's get started...`,
  { header: true },
);

const createMaintenanceWindow = new ScenarioOutput(
  "createMaintenanceWindow",
  "Step 1: Create a Systems Manager maintenance window.",
);

const getMaintenanceWindow = new ScenarioInput(
  "maintenanceWindow",
  "Please enter the maintenance window name:",
  { type: "input", default: defaultMaintenanceWindow },
);
```

```

export const sdkCreateMaintenanceWindow = new ScenarioAction(
  "sdkCreateMaintenanceWindow",
  async (** @type {State} */ state) => {
    try {
      const response = await state.ssmClient.send(
        new CreateMaintenanceWindowCommand({
          Name: state.maintenanceWindow,
          Schedule: "cron(0 10 ? * MON-FRI *)", //The schedule of the maintenance
window in the form of a cron or rate expression.
          Duration: 2, //The duration of the maintenance window in hours.
          Cutoff: 1, //The number of hours before the end of the maintenance window
that Amazon Web Services Systems Manager stops scheduling new tasks for execution.
          AllowUnassociatedTargets: true, //Allow the maintenance window to run on
managed nodes, even if you haven't registered those nodes as targets.
        })),
      );
      state.winId = response.WindowId;
    } catch (caught) {
      console.error(caught.message);
      console.log(
        `An error occurred while creating the maintenance window. Please fix the
error and try again. Error message: ${caught.message}`,
      );
      throw caught;
    }
  },
);

const modifyMaintenanceWindow = new ScenarioOutput(
  "modifyMaintenanceWindow",
  "Modify the maintenance window by changing the schedule.",
);

const sdkModifyMaintenanceWindow = new ScenarioAction(
  "sdkModifyMaintenanceWindow",
  async (** @type {State} */ state) => {
    try {
      await state.ssmClient.send(
        new UpdateMaintenanceWindowCommand({
          WindowId: state.winId,
          Schedule: "cron(0 0 ? * MON *)",
        })),
      );
    } catch (caught) {

```

```

        console.error(caught.message);
        console.log(
            `An error occurred while modifying the maintenance window. Please fix the
            error and try again. Error message: ${caught.message}`,
        );
        throw caught;
    }
},
);

const createSystemsManagerActions = new ScenarioOutput(
    "createSystemsManagerActions",
    "Create a document that defines the actions that Systems Manager performs on your
    EC2 instance.",
);

const getDocumentName = new ScenarioInput(
    "documentName",
    "Please enter the document: ",
    { type: "input", default: defaultDocumentName },
);

const sdkCreateSSMDoc = new ScenarioAction(
    "sdkCreateSSMDoc",
    async (** @type {State} */ state) => {
        const contentData = `{
            "schemaVersion": "2.2",
            "description": "Run a simple shell command",
            "mainSteps": [
                {
                    "action": "aws:runShellScript",
                    "name": "runEchoCommand",
                    "inputs": {
                        "runCommand": [
                            "echo 'Hello, world!'"
                        ]
                    }
                }
            ]
        }`;

        try {
            await state.ssmClient.send(
                new CreateDocumentCommand({
                    Content: contentData,

```

```

        Name: state.documentName,
        DocumentType: "Command",
    })),
    );
} catch (caught) {
    console.log(`Exception type: (${typeof caught})`);
    if (caught instanceof DocumentAlreadyExists) {
        console.log("Document already exists. Continuing...\n");
    } else {
        console.error(caught.message);
        console.log(
            `An error occurred while creating the document. Please fix the error and
            try again. Error message: ${caught.message}`,
        );
        throw caught;
    }
}
},
);

const ec2HelloWorld = new ScenarioOutput(
    "ec2HelloWorld",
    `Now you have the option of running a command on an EC2 instance that echoes
    'Hello, world!'. In order to run this command, you must provide the instance ID
    of a Linux EC2 instance. If you do not already have a running Linux EC2 instance
    in your account, you can create one using the AWS console. For information about
    creating an EC2 instance, see https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-launch-instance-wizard.html.`,
);

const enterIdOrSkipEC2HelloWorld = new ScenarioInput(
    "enterIdOrSkipEC2HelloWorld",
    "Enter your EC2 InstanceId or press enter to skip this step: ",
    { type: "input", default: "" },
);

const sdkEC2HelloWorld = new ScenarioAction(
    "sdkEC2HelloWorld",
    async (/** @type {State} */ state) => {
        try {
            const response = await state.ssmClient.send(
                new SendCommandCommand({
                    DocumentName: state.documentName,
                    InstanceIds: [state.ec2InstanceId],
                })
            );
        } catch (error) {
            console.error("Error sending command:", error);
        }
    })
);
```

```

        TimeoutSeconds: COMMAND_TIMEOUT_DURATION_SECONDS,
      )),
    );
    state.CommandId = response.Command.CommandId;
  } catch (caught) {
    console.error(caught.message);
    console.log(
      `An error occurred while sending the command. Please fix the error and try
again. Error message: ${caught.message}`,
    );
    throw caught;
  }
},
{
  skipWhen: (/** @type {State} */ state) =>
    state.enterIdOrSkipEC2HelloWorld === "",
},
);

const sdkGetCommandTime = new ScenarioAction(
  "sdkGetCommandTime",
  async (/** @type {State} */ state) => {
    const listInvocationsPaginated = [];
    console.log(
      "Let's get the time when the specific command was sent to the specific managed
node.",
    );

    console.log(
      `First, we'll wait for the command to finish executing. This may take up to
${COMMAND_TIMEOUT_DURATION_SECONDS} seconds.`,
    );
    const commandExecutedResult = waitUntilCommandExecuted(
      { client: state.ssmClient },
      {
        CommandId: state.CommandId,
        InstanceId: state.ec2InstanceId,
      },
    );
    // This is necessary because the TimeoutSeconds of SendCommandCommand is only
    for the delivery, not execution.
    try {
      await new Promise((_, reject) =>
        setTimeout(

```

```

        reject,
        COMMAND_TIMEOUT_DURATION_SECONDS * 1000,
        new Error("Command Timed Out"),
    ),
);
} catch (caught) {
    if (caught.message === "Command Timed Out") {
        commandExecutedResult.state = "TIMED_OUT";
    } else {
        throw caught;
    }
}

if (commandExecutedResult.state !== "SUCCESS") {
    console.log(
        `The command with id: ${state.CommandId} did not execute in the allotted
time. Canceling command.`,
    );
    state.ssmClient.send(
        new CancelCommandCommand({
            CommandId: state.CommandId,
        }),
    );
    state.enterIdOrSkipEC2HelloWorld === "";
    return;
}

for await (const page of paginateListCommandInvocations(
    { client: state.ssmClient },
    { CommandId: state.CommandId },
)) {
    listInvocationsPaginated.push(...page.CommandInvocations);
}
/**
 * @type {import('@aws-sdk/client-ssm').CommandInvocation}
 */
const commandInvocation = listInvocationsPaginated.shift(); // Because the call
was made with CommandId, there's only one result, so shift it off.
state.requestedDateTime = commandInvocation.RequestedDateTime;

console.log(
    `The command invocation happened at: ${state.requestedDateTime}.`,
);
},

```

```

{
  skipWhen: (/** @type {State} */ state) =>
    state.enterIdOrSkipEC2HelloWorld === "",
},
);

const createSSMOpsItem = new ScenarioOutput(
  "createSSMOpsItem",
  `Now we will create a Systems Manager OpsItem. An OpsItem is a feature provided by
the Systems Manager service. It is a type of operational data item that allows you
to manage and track various operational issues, events, or tasks within your AWS
environment.
You can create OpsItems to track and manage operational issues as they arise. For
example, you could create an OpsItem whenever your application detects a critical
error or an anomaly in your infrastructure.`,
);

const sdkCreateSSMOpsItem = new ScenarioAction(
  "sdkCreateSSMOpsItem",
  async (/** @type {State} */ state) => {
    try {
      const response = await state.ssmClient.send(
        new CreateOpsItemCommand({
          Description: "Created by the System Manager Javascript API",
          Title: "Disk Space Alert",
          Source: "EC2",
          Category: "Performance",
          Severity: "2",
        })),
    );
    state.opsItemId = response.OpsItemId;
  } catch (caught) {
    console.error(caught.message);
    console.log(
      `An error occurred while creating the ops item. Please fix the error and try
again. Error message: ${caught.message}`,
    );
    throw caught;
  }
},
);

const updateOpsItem = new ScenarioOutput(
  "updateOpsItem",

```

```
(/** @type {State} */ state) =>
  `Now we will update the OpsItem: ${state.opsItemId}`,
);

const sdkUpdateOpsItem = new ScenarioAction(
  "sdkUpdateOpsItem",
  async (/** @type {State} */ state) => {
    try {
      const _response = await state.ssmClient.send(
        new UpdateOpsItemCommand({
          OpsItemId: state.opsItemId,
          Description: `An update to ${state.opsItemId}`,
        }),
      );
    } catch (caught) {
      console.error(caught.message);
      console.log(
        `An error occurred while updating the ops item. Please fix the error and try
again. Error message: ${caught.message}`,
      );
      throw caught;
    }
  },
);

const getOpsItemStatus = new ScenarioOutput(
  "getOpsItemStatus",
  (/** @type {State} */ state) =>
    `Now we will get the status of the OpsItem: ${state.opsItemId}`,
);

const sdkOpsItemStatus = new ScenarioAction(
  "sdkGetOpsItemStatus",
  async (/** @type {State} */ state) => {
    try {
      const response = await state.ssmClient.send(
        new DescribeOpsItemsCommand({
          OpsItemId: state.opsItemId,
        }),
      );
      state.opsItemStatus = response.OpsItemStatus;
    } catch (caught) {
      console.error(caught.message);
      console.log(
```



```

        `An error occurred while describing the ops item. Please fix the error and
        try again. Error message: ${caught.message}`,
    );
    throw caught;
}
},
);

const resolveOpsItem = new ScenarioOutput(
    "resolveOpsItem",
    (/** @type {State} */ state) =>
        `Now we will resolve the OpsItem: ${state.opsItemId}`,
);

const sdkResolveOpsItem = new ScenarioAction(
    "sdkResolveOpsItem",
    async (/** @type {State} */ state) => {
        try {
            const _response = await state.ssmClient.send(
                new UpdateOpsItemCommand({
                    OpsItemId: state.opsItemId,
                    Status: OpsItemStatus.RESOLVED,
                }),
            );
        } catch (caught) {
            console.error(caught.message);
            console.log(
                `An error occurred while updating the ops item. Please fix the error and try
                again. Error message: ${caught.message}`,
            );
            throw caught;
        }
    },
);

const askToDeleteResources = new ScenarioInput(
    "askToDeleteResources",
    "Would you like to delete the Systems Manager resources created during this
    example run?",
    { type: "confirm" },
);

const confirmDeleteChoice = new ScenarioOutput(
    "confirmDeleteChoice",

```

```
(/** @type {State} */ state) => {
  if (state.askToDeleteResources) {
    return "You chose to delete the resources.";
  }
  return "The Systems Manager resources will not be deleted. Please delete them manually to avoid charges.";
},
);

export const sdkDeleteResources = new ScenarioAction(
  "sdkDeleteResources",
  async (/** @type {State} */ state) => {
    try {
      await state.ssmClient.send(
        new DeleteOpsItemCommand({
          OpsItemId: state.opsItemId,
        }),
      );
      console.log(`The ops item: ${state.opsItemId} was successfully deleted.`);
    } catch (caught) {
      console.log(
        `There was a problem deleting the ops item: ${state.opsItemId}. Please delete it manually. Error: ${caught.message}`,
      );
    }

    try {
      await state.ssmClient.send(
        new DeleteMaintenanceWindowCommand({
          Name: state.maintenanceWindow,
          WindowId: state.winId,
        }),
      );
      console.log(
        `The maintenance window: ${state.maintenanceWindow} was successfully deleted.`,
      );
    } catch (caught) {
      console.log(
        `There was a problem deleting the maintenance window: ${state.opsItemId}. Please delete it manually. Error: ${caught.message}`,
      );
    }
  }
});
```

```
    try {
      await state.ssmClient.send(
        new DeleteDocumentCommand({
          Name: state.documentName,
        }),
      );
      console.log(
        `The document: ${state.documentName} was successfully deleted.`
      );
    } catch (caught) {
      console.log(
        `There was a problem deleting the document: ${state.documentName}. Please delete it manually. Error: ${caught.message}`
      );
    }
  },
  { skipWhen: (/** @type {} */ state) => !state.askToDeleteResources },
);

const goodbye = new ScenarioOutput(
  "goodbye",
  "This concludes the Systems Manager Basics scenario for the AWS Javascript SDK v3. Thank you!",
);

const myScenario = new Scenario(
  "SSM Basics",
  [
    greet,
    pressEnter,
    createMaintenanceWindow,
    getMaintenanceWindow,
    sdkCreateMaintenanceWindow,
    modifyMaintenanceWindow,
    pressEnter,
    sdkModifyMaintenanceWindow,
    createSystemsManagerActions,
    getDocumentName,
    sdkCreateSSMDoc,
    ec2HelloWorld,
    enterIdOrSkipEC2HelloWorld,
    sdkEC2HelloWorld,
    sdkGetCommandTime,
    pressEnter,
```

```

    createSSMOpsItem,
    pressEnter,
    sdkCreateSSMOpsItem,
    updateOpsItem,
    pressEnter,
    sdkUpdateOpsItem,
    getOpsItemStatus,
    pressEnter,
    sdkOpsItemStatus,
    resolveOpsItem,
    pressEnter,
    sdkResolveOpsItem,
    askToDeleteResources,
    confirmDeleteChoice,
    sdkDeleteResources,
    goodbye,
  ],
  { ssmClient: new SSMClient({}) },
);

/** @type {{ stepHandlerOptions: StepHandlerOptions }} */
export const main = async (stepHandlerOptions) => {
  await myScenario.run(stepHandlerOptions);
};

// Invoke main function if this file was run directly.
if (process.argv[1] === fileURLToPath(import.meta.url)) {
  const { values } = parseArgs({
    options: {
      yes: {
        type: "boolean",
        short: "y",
      },
    },
  });
  main({ confirmAll: values.yes });
}

```

- Weitere API-Informationen finden Sie in den folgenden Themen der AWS SDK für JavaScript - API-Referenz.
 - [CreateDocument](#)

- [CreateMaintenanceWindow](#)
- [CreateOpsItem](#)
- [DeleteMaintenanceWindow](#)
- [ListCommandInvocations](#)
- [SendCommand](#)
- [UpdateOpsItem](#)

Aktionen

CreateDocument

Das folgende Codebeispiel zeigt die Verwendung `CreateDocument`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { CreateDocumentCommand, SSMClient } from "@aws-sdk/client-ssm";
import { parseArgs } from "node:util";

/**
 * Create an SSM document.
 * @param {{ content: string, name: string, documentType?: DocumentType }}
 */
export const main = async ({ content, name, documentType }) => {
  const client = new SSMClient({});
  try {
    const { documentDescription } = await client.send(
      new CreateDocumentCommand({
        Content: content, // The content for the new SSM document. The content must
                          // not exceed 64KB.
        Name: name,
        DocumentType: documentType, // Document format type can be JSON, YAML, or
        // TEXT. The default format is JSON.
      })
    );
  }
}
```

```

    );
    console.log("Document created successfully.");
    return { DocumentDescription: documentDescription };
  } catch (caught) {
    if (caught instanceof Error && caught.name === "DocumentAlreadyExists") {
      console.warn(`${caught.message}. Did you provide a new document name?`);
    } else {
      throw caught;
    }
  }
};

```

- Einzelheiten zur API finden Sie [CreateDocument](#) in der AWS SDK für JavaScript API-Referenz.

CreateMaintenanceWindow

Das folgende Codebeispiel zeigt die Verwendung `CreateMaintenanceWindow`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```

import { CreateMaintenanceWindowCommand, SSMClient } from "@aws-sdk/client-ssm";
import { parseArgs } from "node:util";

/**
 * Create an SSM maintenance window.
 * @param {{ name: string, allowUnassociatedTargets: boolean, duration: number,
 * cutoff: number, schedule: string, description?: string }}
 */
export const main = async ({
  name,
  allowUnassociatedTargets, // Allow the maintenance window to run on managed nodes,
  even if you haven't registered those nodes as targets.
  duration, // The duration of the maintenance window in hours.
  cutoff, // The number of hours before the end of the maintenance window that
  Amazon Web Services Systems Manager stops scheduling new tasks for execution.

```

```

    schedule, // The schedule of the maintenance window in the form of a cron or rate
    expression.
    description = undefined,
  }) => {
    const client = new SSMClient({});

    try {
      const { windowId } = await client.send(
        new CreateMaintenanceWindowCommand({
          Name: name,
          Description: description,
          AllowUnassociatedTargets: allowUnassociatedTargets, // Allow the maintenance
window to run on managed nodes, even if you haven't registered those nodes as
targets.
          Duration: duration, // The duration of the maintenance window in hours.
          Cutoff: cutoff, // The number of hours before the end of the maintenance
window that Amazon Web Services Systems Manager stops scheduling new tasks for
execution.
          Schedule: schedule, // The schedule of the maintenance window in the form of
a cron or rate expression.
        })),
      );
      console.log(`Maintenance window created with Id: ${windowId}`);
      return { WindowId: windowId };
    } catch (caught) {
      if (caught instanceof Error && caught.name === "MissingParameter") {
        console.warn(`${caught.message}. Did you provide these values?`);
      } else {
        throw caught;
      }
    }
  }
};

```

- Einzelheiten zur API finden Sie [CreateMaintenanceWindow](#) in der AWS SDK für JavaScript API-Referenz.

CreateOpsItem

Das folgende Codebeispiel zeigt die Verwendung `CreateOpsItem`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { CreateOpsItemCommand, SSMClient } from "@aws-sdk/client-ssm";
import { parseArgs } from "node:util";

/**
 * Create an SSM OpsItem.
 * @param {{ title: string, source: string, category?: string, severity?: string }}
 */
export const main = async ({
  title,
  source,
  category = undefined,
  severity = undefined,
}) => {
  const client = new SSMClient({});
  try {
    const { opsItemArn, opsItemId } = await client.send(
      new CreateOpsItemCommand({
        Title: title,
        Source: source, // The origin of the OpsItem, such as Amazon EC2 or Systems
        Manager:
          Category: category,
          Severity: severity,
      })
    );
    console.log(`Ops item created with id: ${opsItemId}`);
    return { OpsItemArn: opsItemArn, OpsItemId: opsItemId };
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MissingParameter") {
      console.warn(`${caught.message}. Did you provide these values?`);
    } else {
      throw caught;
    }
  }
};
```


- Einzelheiten zur API finden Sie [CreateOpsItem](#) in der AWS SDK für JavaScript API-Referenz.

DeleteDocument

Das folgende Codebeispiel zeigt die Verwendung `DeleteDocument`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel](#) einrichten und ausführen.

```
import { DeleteDocumentCommand, SSMClient } from "@aws-sdk/client-ssm";
import { parseArgs } from "node:util";

/**
 * Delete an SSM document.
 * @param {{ documentName: string }}
 */
export const main = async ({ documentName }) => {
  const client = new SSMClient({});
  try {
    await client.send(new DeleteDocumentCommand({ Name: documentName }));
    console.log(`Document '${documentName}' deleted.`);
    return { Deleted: true };
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MissingParameter") {
      console.warn(`${caught.message}. Did you provide this value?`);
    } else {
      throw caught;
    }
  }
};
```

- Einzelheiten zur API finden Sie [DeleteDocument](#) in der AWS SDK für JavaScript API-Referenz.

DeleteMaintenanceWindow

Das folgende Codebeispiel zeigt die Verwendung `DeleteMaintenanceWindow`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { DeleteMaintenanceWindowCommand, SSMClient } from "@aws-sdk/client-ssm";
import { parseArgs } from "node:util";

/**
 * Delete an SSM maintenance window.
 * @param {{ windowId: string }}
 */
export const main = async ({ windowId }) => {
  const client = new SSMClient({});
  try {
    await client.send(
      new DeleteMaintenanceWindowCommand({ WindowId: windowId }),
    );
    console.log(`Maintenance window '${windowId}' deleted.`);
    return { Deleted: true };
  } catch (caught) {
    if (caught instanceof Error && caught.name === "MissingParameter") {
      console.warn(`${caught.message}. Did you provide this value?`);
    } else {
      throw caught;
    }
  }
};
```

- Einzelheiten zur API finden Sie [DeleteMaintenanceWindow](#) in der AWS SDK für JavaScript API-Referenz.

DescribeOpsItems

Das folgende Codebeispiel zeigt die Verwendung `DescribeOpsItems`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import {
  OpsItemFilterOperator,
  OpsItemFilterKey,
  paginateDescribeOpsItems,
  SSMClient,
} from "@aws-sdk/client-ssm";
import { parseArgs } from "node:util";

/**
 * Describe SSM OpsItems.
 * @param {{ opsItemId: string }}
 */
export const main = async ({ opsItemId }) => {
  const client = new SSMClient({});
  try {
    const describeOpsItemsPaginated = [];
    for await (const page of paginateDescribeOpsItems(
      { client },
      {
        OpsItemFilters: {
          Key: OpsItemFilterKey.OPSITEM_ID,
          Operator: OpsItemFilterOperator.EQUAL,
          Values: opsItemId,
        },
      },
    )) {
      describeOpsItemsPaginated.push(...page.OpsItemSummaries);
    }
    console.log("Here are the ops items:");
    console.log(describeOpsItemsPaginated);
    return { OpsItemSummaries: describeOpsItemsPaginated };
  }
}
```

```

    } catch (caught) {
      if (caught instanceof Error && caught.name === "MissingParameter") {
        console.warn(`${caught.message}. Did you provide this value?`);
      }
      throw caught;
    }
  };
};

```

- Einzelheiten zur API finden Sie [DescribeOpsItems](#) in der AWS SDK für JavaScript API-Referenz.

ListCommandInvocations

Das folgende Codebeispiel zeigt die Verwendung `ListCommandInvocations`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```

import { paginateListCommandInvocations, SSMClient } from "@aws-sdk/client-ssm";
import { parseArgs } from "node:util";

/**
 * List SSM command invocations on an instance.
 * @param {{ instanceId: string }}
 */
export const main = async ({ instanceId }) => {
  const client = new SSMClient({});
  try {
    const listCommandInvocationsPaginated = [];
    // The paginate function is a wrapper around the base command.
    const paginator = paginateListCommandInvocations(
      { client },
      {
        InstanceId: instanceId,
      },
    ),

```

```

    );
    for await (const page of paginator) {
        listCommandInvocationsPaginated.push(...page.CommandInvocations);
    }
    console.log("Here is the list of command invocations:");
    console.log(listCommandInvocationsPaginated);
    return { CommandInvocations: listCommandInvocationsPaginated };
} catch (caught) {
    if (caught instanceof Error && caught.name === "ValidationError") {
        console.warn(`${caught.message}. Did you provide a valid instance ID?`);
    }
    throw caught;
}
};

```

- Einzelheiten zur API finden Sie [ListCommandInvocations](#) in der AWS SDK für JavaScript API-Referenz.

SendCommand

Das folgende Codebeispiel zeigt die Verwendung `SendCommand`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```

import { SendCommandCommand, SSMClient } from "@aws-sdk/client-ssm";
import { parseArgs } from "node:util";

/**
 * Send an SSM command to a managed node.
 * @param {{ documentName: string }}
 */
export const main = async ({ documentName }) => {
    const client = new SSMClient({});
    try {

```

```

    await client.send(
      new SendCommandCommand({
        DocumentName: documentName,
      }),
    );
    console.log("Command sent successfully.");
    return { Success: true };
  } catch (caught) {
    if (caught instanceof Error && caught.name === "ValidationError") {
      console.warn(`${caught.message}. Did you provide a valid document name?`);
    } else {
      throw caught;
    }
  }
};

```

- Einzelheiten zur API finden Sie [SendCommand](#) in der AWS SDK für JavaScript API-Referenz.

UpdateMaintenanceWindow

Das folgende Codebeispiel zeigt die Verwendung `UpdateMaintenanceWindow`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```

import { UpdateMaintenanceWindowCommand, SSMClient } from "@aws-sdk/client-ssm";
import { parseArgs } from "node:util";

/**
 * Update an SSM maintenance window.
 * @param {{ windowId: string, allowUnassociatedTargets?: boolean, duration?:
 * number, enabled?: boolean, name?: string, schedule?: string }}
 */
export const main = async ({
  windowId,

```

```

    allowUnassociatedTargets = undefined, //Allow the maintenance window to run on
    managed nodes, even if you haven't registered those nodes as targets.
    duration = undefined, //The duration of the maintenance window in hours.
    enabled = undefined,
    name = undefined,
    schedule = undefined, //The schedule of the maintenance window in the form of a
    cron or rate expression.
  }) => {
    const client = new SSMClient({});
    try {
      const { opsItemArn, opsItemId } = await client.send(
        new UpdateMaintenanceWindowCommand({
          WindowId: windowId,
          AllowUnassociatedTargets: allowUnassociatedTargets,
          Duration: duration,
          Enabled: enabled,
          Name: name,
          Schedule: schedule,
        })),
      );
      console.log("Maintenance window updated.");
      return { OpsItemArn: opsItemArn, OpsItemId: opsItemId };
    } catch (caught) {
      if (caught instanceof Error && caught.name === "ValidationError") {
        console.warn(`${caught.message}. Are these values correct?`);
      } else {
        throw caught;
      }
    }
  };
};

```

- Einzelheiten zur API finden Sie [UpdateMaintenanceWindow](#) in der AWS SDK für JavaScript API-Referenz.

UpdateOpsItem

Das folgende Codebeispiel zeigt die Verwendung `UpdateOpsItem`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

```
import { UpdateOpsItemCommand, SSMClient } from "@aws-sdk/client-ssm";
import { parseArgs } from "node:util";

/**
 * Update an SSM OpsItem.
 * @param {{ opsItemId: string, status?: OpsItemStatus }}
 */
export const main = async ({
  opsItemId,
  status = undefined, // The OpsItem status. Status can be Open, In Progress, or Resolved
}) => {
  const client = new SSMClient({});
  try {
    await client.send(
      new UpdateOpsItemCommand({
        OpsItemId: opsItemId,
        Status: status,
      }),
    );
    console.log("Ops item updated.");
    return { Success: true };
  } catch (caught) {
    if (
      caught instanceof Error &&
      caught.name === "OpsItemLimitExceededException"
    ) {
      console.warn(
        `Couldn't create ops item because you have exceeded your open OpsItem limit. ${caught.message}.`,
      );
    } else {
      throw caught;
    }
  }
}
```



```
}  
};
```

- Einzelheiten zur API finden Sie [UpdateOpsItem](#) in der AWS SDK für JavaScript API-Referenz.

Amazon Textract Textract-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK für JavaScript (v3) mit Amazon Textract Aktionen ausführen und allgemeine Szenarien implementieren.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb eines Service aufrufen oder mit anderen AWS-Services kombinieren.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kodex finden.

Themen

- [Szenarien](#)

Szenarien

Erstellen Sie eine Amazon-Textract-Explorer-Anwendung

Die folgenden Codebeispiele zeigen, wie die Amazon-Textract-Ausgabe in einer interaktiven Anwendung untersucht werden kann.

SDK für JavaScript (v3)

Zeigt, wie Sie mit AWS SDK für JavaScript eine React-Anwendung erstellen, die Amazon Textract verwendet, um Daten aus einem Dokumentbild zu extrahieren und auf einer interaktiven Webseite anzuzeigen. Dieses Beispiel wird in einem Webbrowser ausgeführt und erfordert eine authentifizierte Amazon-Cognito-Identität für Anmeldeinformationen. Es verwendet Amazon Simple Storage Service (Amazon S3) zur Speicherung und fragt für Benachrichtigungen eine Amazon Simple Queue Service (Amazon SQS)-Warteschlange ab, die ein Amazon Simple Notification Service (Amazon SNS)-Thema abonniert hat.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

In diesem Beispiel verwendete Dienste

- Amazon Cognito Identity
- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon Textract

Erstellen einer Anwendung zum Analysieren von Kundenfeedback

Das folgende Codebeispiel zeigt, wie Sie eine Anwendung erstellen, die Kundenkommentarkarten analysiert, sie aus der ursprünglichen Sprache übersetzt, die Stimmung ermittelt und auf der Grundlage des übersetzten Texts eine Audiodatei generiert.

SDK für JavaScript (v3)

Diese Beispielanwendung analysiert und speichert Kundenfeedback-Karten. Sie ist auf die Anforderungen eines fiktiven Hotels in New York City zugeschnitten. Das Hotel erhält Feedback von Gästen in Form von physischen Kommentarkarten in verschiedenen Sprachen. Dieses Feedback wird über einen Webclient in die App hochgeladen. Nachdem ein Bild einer Kommentarkarte hochgeladen wurde, werden folgende Schritte ausgeführt:

- Der Text wird mithilfe von Amazon Textract aus dem Bild extrahiert.
- Amazon Comprehend ermittelt die Stimmung und die Sprache des extrahierten Textes.
- Der extrahierte Text wird mithilfe von Amazon Translate ins Englische übersetzt.
- Amazon Polly generiert auf der Grundlage des extrahierten Texts eine Audiodatei.

Die vollständige App kann mithilfe des AWS CDK bereitgestellt werden. Quellcode und Anweisungen zur Bereitstellung finden Sie im Projekt unter [GitHub](#). Die folgenden Auszüge zeigen, wie der innerhalb von Lambda-Funktionen verwendet AWS SDK für JavaScript wird.

```
import {
  ComprehendClient,
  DetectDominantLanguageCommand,
  DetectSentimentCommand,
} from "@aws-sdk/client-comprehend";

/**
 * Determine the language and sentiment of the extracted text.
```

```

*
* @param {{ source_text: string }} extractTextOutput
*/
export const handler = async (extractTextOutput) => {
  const comprehendClient = new ComprehendClient({});

  const detectDominantLanguageCommand = new DetectDominantLanguageCommand({
    Text: extractTextOutput.source_text,
  });

  // The source language is required for sentiment analysis and
  // translation in the next step.
  const { Languages } = await comprehendClient.send(
    detectDominantLanguageCommand,
  );

  const languageCode = Languages[0].LanguageCode;

  const detectSentimentCommand = new DetectSentimentCommand({
    Text: extractTextOutput.source_text,
    LanguageCode: languageCode,
  });

  const { Sentiment } = await comprehendClient.send(detectSentimentCommand);

  return {
    sentiment: Sentiment,
    language_code: languageCode,
  };
};

```

```

import {
  DetectDocumentTextCommand,
  TextractClient,
} from "@aws-sdk/client-textract";

/**
 * Fetch the S3 object from the event and analyze it using Amazon Textract.
 *
 * @param {import("@types/aws-lambda").EventBridgeEvent<"Object Created">}
  eventBridgeS3Event
 */
export const handler = async (eventBridgeS3Event) => {

```

```

const textractClient = new TextractClient();

const detectDocumentTextCommand = new DetectDocumentTextCommand({
  Document: {
    S3Object: {
      Bucket: eventBridgeS3Event.bucket,
      Name: eventBridgeS3Event.object,
    },
  },
});

// Textract returns a list of blocks. A block can be a line, a page, word, etc.
// Each block also contains geometry of the detected text.
// For more information on the Block type, see https://docs.aws.amazon.com/textract/latest/dg/API\_Block.html.
const { Blocks } = await textractClient.send(detectDocumentTextCommand);

// For the purpose of this example, we are only interested in words.
const extractedWords = Blocks.filter((b) => b.BlockType === "WORD").map(
  (b) => b.Text,
);

return extractedWords.join(" ");
};

```

```

import { PollyClient, SynthesizeSpeechCommand } from "@aws-sdk/client-polly";
import { S3Client } from "@aws-sdk/client-s3";
import { Upload } from "@aws-sdk/lib-storage";

/**
 * Synthesize an audio file from text.
 *
 * @param {{ bucket: string, translated_text: string, object: string }}
 * sourceDestinationConfig
 */
export const handler = async (sourceDestinationConfig) => {
  const pollyClient = new PollyClient({});

  const synthesizeSpeechCommand = new SynthesizeSpeechCommand({
    Engine: "neural",
    Text: sourceDestinationConfig.translated_text,
    VoiceId: "Ruth",
    OutputFormat: "mp3",
  });

```

```
});

const { AudioStream } = await pollyClient.send(synthesizeSpeechCommand);

const audioKey = `${sourceDestinationConfig.object}.mp3`;

// Store the audio file in S3.
const s3Client = new S3Client();
const upload = new Upload({
  client: s3Client,
  params: {
    Bucket: sourceDestinationConfig.bucket,
    Key: audioKey,
    Body: AudioStream,
    ContentType: "audio/mp3",
  },
});

await upload.done();
return audioKey;
};
```

```
import {
  TranslateClient,
  TranslateTextCommand,
} from "@aws-sdk/client-translate";

/**
 * Translate the extracted text to English.
 *
 * @param {{ extracted_text: string, source_language_code: string }}
  textAndSourceLanguage
 */
export const handler = async (textAndSourceLanguage) => {
  const translateClient = new TranslateClient({});

  const translateCommand = new TranslateTextCommand({
    SourceLanguageCode: textAndSourceLanguage.source_language_code,
    TargetLanguageCode: "en",
    Text: textAndSourceLanguage.extracted_text,
  });

  const { TranslatedText } = await translateClient.send(translateCommand);
```

```
    return { translated_text: TranslatedText };  
};
```

In diesem Beispiel verwendete Dienste

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

Amazon Transcribe Transcribe-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK für JavaScript (v3) mit Amazon Transcribe Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarien anzeigen.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb eines Service aufrufen oder mit anderen AWS-Services kombinieren.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kodex finden.

Themen

- [Aktionen](#)
- [Szenarien](#)

Aktionen

DeleteMedicalTranscriptionJob

Das folgende Codebeispiel zeigt, wie Sie es verwenden. DeleteMedicalTranscriptionJob

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie den Client.

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

Einen Auftrag für medizinische Transkription löschen.

```
// Import the required AWS SDK clients and commands for Node.js
import { DeleteMedicalTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "../libs/transcribeClient.js";

// Set the parameters
export const params = {
  MedicalTranscriptionJobName: "MEDICAL_JOB_NAME", // For example,
  'medical_transcription_demo'
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new DeleteMedicalTranscriptionJobCommand(params),
    );
    console.log("Success - deleted");
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};

run();
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [DeleteMedicalTranscriptionJob](#) in der AWS SDK für JavaScript API-Referenz.

DeleteTranscriptionJob

Das folgende Codebeispiel zeigt die Verwendung `DeleteTranscriptionJob`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel](#) einrichten und ausführen.

Löschen eines Transkriptionsauftrags.

```
// Import the required AWS SDK clients and commands for Node.js
import { DeleteTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "../libs/transcribeClient.js";

// Set the parameters
export const params = {
  TranscriptionJobName: "JOB_NAME", // Required. For example, 'transcription_demo'
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new DeleteTranscriptionJobCommand(params),
    );
    console.log("Success - deleted");
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};
```



```
run();
```

Erstellen Sie den Client.

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";  
// Set the AWS Region.  
const REGION = "REGION"; //e.g. "us-east-1"  
// Create an Amazon Transcribe service client object.  
const transcribeClient = new TranscribeClient({ region: REGION });  
export { transcribeClient };
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [DeleteTranscriptionJob](#) in der AWS SDK für JavaScript API-Referenz.

ListMedicalTranscriptionJobs

Das folgende Codebeispiel zeigt die Verwendung `ListMedicalTranscriptionJobs`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie den Client.

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";  
// Set the AWS Region.  
const REGION = "REGION"; //e.g. "us-east-1"  
// Create an Amazon Transcribe service client object.  
const transcribeClient = new TranscribeClient({ region: REGION });  
export { transcribeClient };
```

Auflisten medizinischer Transkriptionsjobs.

```
// Import the required AWS SDK clients and commands for Node.js
import { StartMedicalTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "../libs/transcribeClient.js";

// Set the parameters
export const params = {
  MedicalTranscriptionJobName: "MEDICAL_JOB_NAME", // Required
  OutputBucketName: "OUTPUT_BUCKET_NAME", // Required
  Specialty: "PRIMARYCARE", // Required. Possible values are 'PRIMARYCARE'
  Type: "JOB_TYPE", // Required. Possible values are 'CONVERSATION' and 'DICTATION'
  LanguageCode: "LANGUAGE_CODE", // For example, 'en-US'
  MediaFormat: "SOURCE_FILE_FORMAT", // For example, 'wav'
  Media: {
    MediaFileUri: "SOURCE_FILE_LOCATION",
    // The S3 object location of the input media file. The URI must be in the same
    // region
    // as the API endpoint that you are calling. For example,
    // "https://transcribe-demo.s3-REGION.amazonaws.com/hello_world.wav"
  },
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new StartMedicalTranscriptionJobCommand(params),
    );
    console.log("Success - put", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};

run();
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [ListMedicalTranscriptionJobs](#) in der AWS SDK für JavaScript API-Referenz.

ListTranscriptionJobs

Das folgende Codebeispiel zeigt die Verwendung `ListTranscriptionJobs`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu [GitHub](#). Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Auflisten von Transkriptionsaufträgen.

```
// Import the required AWS SDK clients and commands for Node.js

import { ListTranscriptionJobsCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "../libs/transcribeClient.js";

// Set the parameters
export const params = {
  JobNameContains: "KEYWORD", // Not required. Returns only transcription
  // job names containing this string
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new ListTranscriptionJobsCommand(params),
    );
    console.log("Success", data.TranscriptionJobSummaries);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};

run();
```

Erstellen Sie den Client.

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
```

```
// Set the AWS Region.  
const REGION = "REGION"; //e.g. "us-east-1"  
// Create an Amazon Transcribe service client object.  
const transcribeClient = new TranscribeClient({ region: REGION });  
export { transcribeClient };
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [ListTranscriptionJobs](#) in der AWS SDK für JavaScript API-Referenz.

StartMedicalTranscriptionJob

Das folgende Codebeispiel zeigt die Verwendung `StartMedicalTranscriptionJob`.

SDK für JavaScript (v3)

Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Erstellen Sie den Client.

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";  
// Set the AWS Region.  
const REGION = "REGION"; //e.g. "us-east-1"  
// Create an Amazon Transcribe service client object.  
const transcribeClient = new TranscribeClient({ region: REGION });  
export { transcribeClient };
```

Einen Auftrag für medizinische Transkription starten.

```
// Import the required AWS SDK clients and commands for Node.js  
import { StartMedicalTranscriptionJobCommand } from "@aws-sdk/client-transcribe";  
import { transcribeClient } from "../libs/transcribeClient.js";  
  
// Set the parameters
```

```
export const params = {
  MedicalTranscriptionJobName: "MEDICAL_JOB_NAME", // Required
  OutputBucketName: "OUTPUT_BUCKET_NAME", // Required
  Specialty: "PRIMARYCARE", // Required. Possible values are 'PRIMARYCARE'
  Type: "JOB_TYPE", // Required. Possible values are 'CONVERSATION' and 'DICTATION'
  LanguageCode: "LANGUAGE_CODE", // For example, 'en-US'
  MediaFormat: "SOURCE_FILE_FORMAT", // For example, 'wav'
  Media: {
    MediaFileUri: "SOURCE_FILE_LOCATION",
    // The S3 object location of the input media file. The URI must be in the same
    // region
    // as the API endpoint that you are calling. For example,
    // "https://transcribe-demo.s3-REGION.amazonaws.com/hello_world.wav"
  },
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new StartMedicalTranscriptionJobCommand(params),
    );
    console.log("Success - put", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};

run();
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [StartMedicalTranscriptionJob](#) in der AWS SDK für JavaScript API-Referenz.

StartTranscriptionJob

Das folgende Codebeispiel zeigt die Verwendung `StartTranscriptionJob`.

SDK für JavaScript (v3)

 Note

Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das [AWS -Code-Beispiel-](#) einrichten und ausführen.

Einen Transkriptionsauftrag starten.

```
// Import the required AWS SDK clients and commands for Node.js
import { StartTranscriptionJobCommand } from "@aws-sdk/client-transcribe";
import { transcribeClient } from "../libs/transcribeClient.js";

// Set the parameters
export const params = {
  TranscriptionJobName: "JOB_NAME",
  LanguageCode: "LANGUAGE_CODE", // For example, 'en-US'
  MediaFormat: "SOURCE_FILE_FORMAT", // For example, 'wav'
  Media: {
    MediaFileUri: "SOURCE_LOCATION",
    // For example, "https://transcribe-demo.s3-REGION.amazonaws.com/
    hello_world.wav"
  },
  OutputBucketName: "OUTPUT_BUCKET_NAME",
};

export const run = async () => {
  try {
    const data = await transcribeClient.send(
      new StartTranscriptionJobCommand(params),
    );
    console.log("Success - put", data);
    return data; // For unit tests.
  } catch (err) {
    console.log("Error", err);
  }
};

run();
```

Erstellen Sie den Client.

```
import { TranscribeClient } from "@aws-sdk/client-transcribe";
// Set the AWS Region.
const REGION = "REGION"; //e.g. "us-east-1"
// Create an Amazon Transcribe service client object.
const transcribeClient = new TranscribeClient({ region: REGION });
export { transcribeClient };
```

- Weitere Informationen finden Sie im [AWS SDK für JavaScript -Entwicklerhandbuch](#).
- Einzelheiten zur API finden Sie [StartTranscriptionJob](#) in der AWS SDK für JavaScript API-Referenz.

Szenarien

Erstellen einer Amazon-Transcribe-Streaming-App

Das folgende Code-Beispiel zeigt, wie Sie eine App erstellen, die Live-Audio in Echtzeit aufzeichnet, transkribiert und übersetzt und die Ergebnisse per E-Mail sendet.

SDK für JavaScript (v3)

Zeigt, wie Amazon Transcribe verwendet wird, um eine App zu erstellen, die Live-Audio in Echtzeit aufzeichnet, transkribiert und übersetzt und die Ergebnisse mit Amazon Simple Email Service (Amazon SES) per E-Mail sendet.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

In diesem Beispiel verwendete Dienste

- Amazon Comprehend
- Amazon SES
- Amazon Transcribe
- Amazon Translate

Amazon Translate Translate-Beispiele mit SDK für JavaScript (v3)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK für JavaScript (v3) mit Amazon Translate Aktionen ausführen und allgemeine Szenarien implementieren.

Szenarien sind Codebeispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb eines Service aufrufen oder mit anderen AWS-Services kombinieren.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, wo Sie Anweisungen zum Einrichten und Ausführen des Codes im Kodex finden.

Themen

- [Szenarien](#)

Szenarien

Erstellen einer Amazon-Transcribe-Streaming-App

Das folgende Code-Beispiel zeigt, wie Sie eine App erstellen, die Live-Audio in Echtzeit aufzeichnet, transkribiert und übersetzt und die Ergebnisse per E-Mail sendet.

SDK für JavaScript (v3)

Zeigt, wie Amazon Transcribe verwendet wird, um eine App zu erstellen, die Live-Audio in Echtzeit aufzeichnet, transkribiert und übersetzt und die Ergebnisse mit Amazon Simple Email Service (Amazon SES) per E-Mail sendet.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel unter [GitHub](#).

In diesem Beispiel verwendete Dienste

- Amazon Comprehend
- Amazon SES
- Amazon Transcribe
- Amazon Translate

Erstellen eines Amazon-Lex-Chatbots

Das folgende Codebeispiele zeigt, wie Sie einen Chatbot erstellen können, um die Besucher Ihrer Website einzubinden.

SDK für JavaScript (v3)

Zeigt, wie Sie die Amazon-Lex-API verwenden können, um einen Chatbot innerhalb einer Webanwendung zu erstellen, um die Besucher Ihrer Website einzubinden.

Den vollständigen Quellcode und Anweisungen zur Einrichtung und Ausführung finden Sie im vollständigen Beispiel „[Einen Amazon Lex-Chatbot erstellen](#)“ im AWS SDK für JavaScript Entwicklerhandbuch.

In diesem Beispiel verwendete Dienste

- Amazon Comprehend
- Amazon Lex
- Amazon Translate

Erstellen einer Anwendung zum Analysieren von Kundenfeedback

Das folgende Codebeispiel zeigt, wie Sie eine Anwendung erstellen, die Kundenkommentarkarten analysiert, sie aus der ursprünglichen Sprache übersetzt, die Stimmung ermittelt und auf der Grundlage des übersetzten Texts eine Audiodatei generiert.

SDK für JavaScript (v3)

Diese Beispielanwendung analysiert und speichert Kundenfeedback-Karten. Sie ist auf die Anforderungen eines fiktiven Hotels in New York City zugeschnitten. Das Hotel erhält Feedback von Gästen in Form von physischen Kommentarkarten in verschiedenen Sprachen. Dieses Feedback wird über einen Webclient in die App hochgeladen. Nachdem ein Bild einer Kommentarkarte hochgeladen wurde, werden folgende Schritte ausgeführt:

- Der Text wird mithilfe von Amazon Textract aus dem Bild extrahiert.
- Amazon Comprehend ermittelt die Stimmung und die Sprache des extrahierten Textes.
- Der extrahierte Text wird mithilfe von Amazon Translate ins Englische übersetzt.
- Amazon Polly generiert auf der Grundlage des extrahierten Texts eine Audiodatei.

Die vollständige App kann mithilfe des AWS CDK bereitgestellt werden. Quellcode und Anweisungen zur Bereitstellung finden Sie im Projekt unter [GitHub](#). Die folgenden Auszüge zeigen, wie der innerhalb von Lambda-Funktionen verwendet AWS SDK für JavaScript wird.

```
import {
```

```

    ComprehendClient,
    DetectDominantLanguageCommand,
    DetectSentimentCommand,
  } from "@aws-sdk/client-comprehend";

/**
 * Determine the language and sentiment of the extracted text.
 *
 * @param {{ source_text: string }} extractTextOutput
 */
export const handler = async (extractTextOutput) => {
  const comprehendClient = new ComprehendClient({});

  const detectDominantLanguageCommand = new DetectDominantLanguageCommand({
    Text: extractTextOutput.source_text,
  });

  // The source language is required for sentiment analysis and
  // translation in the next step.
  const { Languages } = await comprehendClient.send(
    detectDominantLanguageCommand,
  );

  const languageCode = Languages[0].LanguageCode;

  const detectSentimentCommand = new DetectSentimentCommand({
    Text: extractTextOutput.source_text,
    LanguageCode: languageCode,
  });

  const { Sentiment } = await comprehendClient.send(detectSentimentCommand);

  return {
    sentiment: Sentiment,
    language_code: languageCode,
  };
};

```

```

import {
  DetectDocumentTextCommand,
  TextractClient,
} from "@aws-sdk/client-textract";

```

```

/**
 * Fetch the S3 object from the event and analyze it using Amazon Textract.
 *
 * @param {import("@types/aws-lambda").EventBridgeEvent<"Object Created">}
eventBridgeS3Event
 */
export const handler = async (eventBridgeS3Event) => {
  const textractClient = new TextractClient();

  const detectDocumentTextCommand = new DetectDocumentTextCommand({
    Document: {
      S3Object: {
        Bucket: eventBridgeS3Event.bucket,
        Name: eventBridgeS3Event.object,
      },
    },
  });

  // Textract returns a list of blocks. A block can be a line, a page, word, etc.
  // Each block also contains geometry of the detected text.
  // For more information on the Block type, see https://docs.aws.amazon.com/
  textract/latest/dg/API_Block.html.
  const { Blocks } = await textractClient.send(detectDocumentTextCommand);

  // For the purpose of this example, we are only interested in words.
  const extractedWords = Blocks.filter((b) => b.BlockType === "WORD").map(
    (b) => b.Text,
  );

  return extractedWords.join(" ");
};

```

```

import { PollyClient, SynthesizeSpeechCommand } from "@aws-sdk/client-polly";
import { S3Client } from "@aws-sdk/client-s3";
import { Upload } from "@aws-sdk/lib-storage";

/**
 * Synthesize an audio file from text.
 *
 * @param {{ bucket: string, translated_text: string, object: string }}
sourceDestinationConfig
 */
export const handler = async (sourceDestinationConfig) => {

```

```

const pollyClient = new PollyClient({});

const synthesizeSpeechCommand = new SynthesizeSpeechCommand({
  Engine: "neural",
  Text: sourceDestinationConfig.translated_text,
  VoiceId: "Ruth",
  OutputFormat: "mp3",
});

const { AudioStream } = await pollyClient.send(synthesizeSpeechCommand);

const audioKey = `${sourceDestinationConfig.object}.mp3`;

// Store the audio file in S3.
const s3Client = new S3Client();
const upload = new Upload({
  client: s3Client,
  params: {
    Bucket: sourceDestinationConfig.bucket,
    Key: audioKey,
    Body: AudioStream,
    ContentType: "audio/mp3",
  },
});

await upload.done();
return audioKey;
};

```

```

import {
  TranslateClient,
  TranslateTextCommand,
} from "@aws-sdk/client-translate";

/**
 * Translate the extracted text to English.
 *
 * @param {{ extracted_text: string, source_language_code: string }}
 * textAndSourceLanguage
 */
export const handler = async (textAndSourceLanguage) => {
  const translateClient = new TranslateClient({});

```

```
const translateCommand = new TranslateTextCommand({
  SourceLanguageCode: textAndSourceLanguage.source_language_code,
  TargetLanguageCode: "en",
  Text: textAndSourceLanguage.extracted_text,
});

const { TranslatedText } = await translateClient.send(translateCommand);

return { translated_text: TranslatedText };
};
```

In diesem Beispiel verwendete Dienste

- Amazon Comprehend
- Lambda
- Amazon Polly
- Amazon Textract
- Amazon Translate

Sicherheit für dieses AWS Produkt oder diese Dienstleistung

Cloud-Sicherheit genießt bei Amazon Web Services (AWS) höchste Priorität. Als AWS -Kunde profitieren Sie von einer Rechenzentrums- und Netzwerkarchitektur, die zur Erfüllung der Anforderungen von Organisationen entwickelt wurden, für die Sicherheit eine kritische Bedeutung hat. Sicherheit ist eine gemeinsame Verantwortung zwischen Ihnen AWS und Ihnen. Im [Modell der übergreifenden Verantwortlichkeit](#) wird Folgendes mit „Sicherheit der Cloud“ bzw. „Sicherheit in der Cloud“ umschrieben:

Sicherheit der Cloud — AWS ist verantwortlich für den Schutz der Infrastruktur, auf der alle in der AWS Cloud angebotenen Dienste ausgeführt werden, und für die Bereitstellung von Diensten, die Sie sicher nutzen können. Unsere Sicherheitsverantwortung hat bei uns höchste Priorität AWS, und die Wirksamkeit unserer Sicherheit wird im Rahmen der [AWS Compliance-Programme](#) regelmäßig von externen Prüfern getestet und verifiziert.

Sicherheit in der Cloud — Ihre Verantwortung richtet sich nach dem von Ihnen genutzten AWS Dienst und anderen Faktoren, wie der Sensibilität Ihrer Daten, den Anforderungen Ihres Unternehmens und den geltenden Gesetzen und Vorschriften.

Dieses AWS Produkt oder dieser Service folgt dem [Modell der gemeinsamen Verantwortung](#) in Bezug auf die spezifischen Amazon Web Services (AWS) -Services, die es unterstützt. Informationen zur AWS Servicesicherheit finden Sie auf der [Seite mit der Dokumentation zur AWS Servicesicherheit](#) und den [AWS Services, für die das AWS Compliance-Programm zur Einhaltung der](#) Vorschriften zuständig ist.

Themen

- [Datenschutz in diesem AWS Produkt oder dieser Dienstleistung](#)
- [Identitäts- und Zugriffsverwaltung](#)
- [Überprüfung der Einhaltung der Vorschriften für dieses AWS Produkt oder diese Dienstleistung](#)
- [Ausfallsicherheit für dieses AWS Produkt oder diese Dienstleistung](#)
- [Sicherheit der Infrastruktur für dieses AWS Produkt oder diesen Service](#)
- [Erzwingen Sie eine TLS-Mindestversion](#)

Datenschutz in diesem AWS Produkt oder dieser Dienstleistung

Das AWS [Modell](#) der gilt für den Datenschutz in diesem AWS Produkt oder dieser Dienstleistung. Wie in diesem Modell beschrieben, AWS ist es verantwortlich für den Schutz der globalen Infrastruktur, auf der alle Systeme laufen AWS Cloud. Sie sind dafür verantwortlich, die Kontrolle über Ihre in dieser Infrastruktur gehosteten Inhalte zu behalten. Sie sind auch für die Sicherheitskonfiguration und die Verwaltungsaufgaben für die von Ihnen verwendeten AWS-Services verantwortlich. Weitere Informationen zum Datenschutz finden Sie unter [Häufig gestellte Fragen zum Datenschutz](#). Informationen zum Datenschutz in Europa finden Sie im Blog-Beitrag [AWS -Modell der geteilten Verantwortung und in der DSGVO](#) im AWS -Sicherheitsblog.

Aus Datenschutzgründen empfehlen wir, dass Sie AWS-Konto Anmeldeinformationen schützen und einzelne Benutzer mit AWS IAM Identity Center oder AWS Identity and Access Management (IAM) einrichten. So erhält jeder Benutzer nur die Berechtigungen, die zum Durchführen seiner Aufgaben erforderlich sind. Außerdem empfehlen wir, die Daten mit folgenden Methoden schützen:

- Verwenden Sie für jedes Konto die Multi-Faktor-Authentifizierung (MFA).
- Wird verwendet SSL/TLS , um mit AWS Ressourcen zu kommunizieren. Wir benötigen TLS 1.2 und empfehlen TLS 1.3.
- Richten Sie die API und die Protokollierung von Benutzeraktivitäten mit ein AWS CloudTrail. Informationen zur Verwendung von CloudTrail Pfaden zur Erfassung von AWS Aktivitäten finden Sie unter [Arbeiten mit CloudTrail Pfaden](#) im AWS CloudTrail Benutzerhandbuch.
- Verwenden Sie AWS Verschlüsselungslösungen zusammen mit allen darin enthaltenen Standardsicherheitskontrollen AWS-Services.
- Verwenden Sie erweiterte verwaltete Sicherheitsservices wie Amazon Macie, die dabei helfen, in Amazon S3 gespeicherte persönliche Daten zu erkennen und zu schützen.
- Wenn Sie für den Zugriff AWS über eine Befehlszeilenschnittstelle oder eine API FIPS 140-3-validierte kryptografische Module benötigen, verwenden Sie einen FIPS-Endpunkt. Weitere Informationen über verfügbare FIPS-Endpunkte finden Sie unter [Federal Information Processing Standard \(FIPS\) 140-3](#).

Wir empfehlen dringend, in Freitextfeldern, z. B. im Feld Name, keine vertraulichen oder sensiblen Informationen wie die E-Mail-Adressen Ihrer Kunden einzugeben. Dazu gehört auch, wenn Sie mit diesem AWS Produkt oder Service oder einem anderen AWS-Services über die Konsole, API oder arbeiten. AWS CLI AWS SDKs Alle Daten, die Sie in Tags oder Freitextfelder eingeben, die für Namen verwendet werden, können für Abrechnungs- oder Diagnoseprotokolle verwendet

werden. Wenn Sie eine URL für einen externen Server bereitstellen, empfehlen wir dringend, keine Anmeldeinformationen zur Validierung Ihrer Anforderung an den betreffenden Server in die URL einzuschließen.

Identitäts- und Zugriffsverwaltung

AWS Identity and Access Management (IAM) hilft einem Administrator AWS-Service, den Zugriff auf Ressourcen sicher zu AWS kontrollieren. IAM-Administratoren kontrollieren, wer authentifiziert (angemeldet) und autorisiert werden kann (über Berechtigungen verfügt), um Ressourcen zu verwenden. AWS IAM ist ein Programm AWS-Service, das Sie ohne zusätzliche Kosten nutzen können.

Themen

- [Zielgruppe](#)
- [Authentifizierung mit Identitäten](#)
- [Verwalten des Zugriffs mit Richtlinien](#)
- [Wie AWS-Services arbeiten Sie mit IAM](#)
- [Fehlerbehebung bei AWS Identität und Zugriff](#)

Zielgruppe

Die Art und Weise, wie Sie AWS Identity and Access Management (IAM) verwenden, hängt von der Arbeit ab, in der Sie tätig sind. AWS

Dienstbenutzer — Wenn Sie dies AWS-Services für Ihre Arbeit verwenden, stellt Ihnen Ihr Administrator die erforderlichen Anmeldeinformationen und Berechtigungen zur Verfügung. Wenn Sie für Ihre Arbeit mehr AWS Funktionen verwenden, benötigen Sie möglicherweise zusätzliche Berechtigungen. Wenn Sie die Funktionsweise der Zugriffskontrolle nachvollziehen, wissen Sie bereits, welche Berechtigungen Sie von Ihrem Administrator anfordern müssen. Falls Sie auf eine Funktion nicht zugreifen können AWS, finden [Fehlerbehebung bei AWS Identität und Zugriff](#) Sie weitere Informationen in der Bedienungsanleitung der von AWS-Service Ihnen verwendeten.

Serviceadministrator — Wenn Sie in Ihrem Unternehmen für die AWS Ressourcen verantwortlich sind, haben Sie wahrscheinlich vollen Zugriff auf AWS. Es ist Ihre Aufgabe, zu bestimmen, auf welche AWS Funktionen und Ressourcen Ihre Servicebenutzer zugreifen sollen. Anschließend müssen Sie Anforderungen an Ihren IAM-Administrator senden, um die Berechtigungen der

Servicebenutzer zu ändern. Lesen Sie die Informationen auf dieser Seite, um die Grundkonzepte von IAM nachzuvollziehen. Weitere Informationen darüber, wie Ihr Unternehmen IAM verwenden kann AWS, finden Sie in der Benutzeranleitung des von AWS-Service Ihnen verwendeten.

IAM-Administrator: Wenn Sie als IAM-Administrator fungieren, sollten Sie Einzelheiten dazu kennen, wie Sie Richtlinien zur Verwaltung des Zugriffs auf AWS verfassen können. Beispiele für AWS identitätsbasierte Richtlinien, die Sie in IAM verwenden können, finden Sie im Benutzerhandbuch der AWS-Service von Ihnen verwendeten.

Authentifizierung mit Identitäten

Authentifizierung ist die Art und Weise, wie Sie sich AWS mit Ihren Identitätsdaten anmelden. Sie müssen sich als IAM-Benutzer authentifizieren oder eine IAM-Rolle annehmen. Root-Benutzer des AWS-Kontos

Sie können sich als föderierte Identität anmelden, indem Sie Anmeldeinformationen aus einer Identitätsquelle wie AWS IAM Identity Center (IAM Identity Center), Single Sign-On-Authentifizierung oder Anmeldeinformationen verwenden. Google/Facebook Weitere Informationen zum Anmelden finden Sie unter [So melden Sie sich bei Ihrem AWS-Konto an](#) im Benutzerhandbuch für AWS-Anmeldung .

AWS Bietet für den programmatischen Zugriff ein SDK und eine CLI zum kryptografischen Signieren von Anfragen. Weitere Informationen finden Sie unter [AWS Signature Version 4 for API requests](#) im IAM-Benutzerhandbuch.

AWS-Konto Root-Benutzer

Wenn Sie einen erstellen AWS-Konto, beginnen Sie mit einer Anmeldeidentität, dem sogenannten AWS-Konto Root-Benutzer, der vollständigen Zugriff auf alle AWS-Services Ressourcen hat. Wir raten ausdrücklich davon ab, den Root-Benutzer für Alltagsaufgaben zu verwenden. Eine Liste der Aufgaben, für die Sie sich als Root-Benutzer anmelden müssen, finden Sie unter [Tasks that require root user credentials](#) im IAM-Benutzerhandbuch.

Verbundidentität

Es hat sich bewährt, dass menschliche Benutzer für den Zugriff AWS-Services mithilfe temporärer Anmeldeinformationen einen Verbund mit einem Identitätsanbieter verwenden müssen.

Eine föderierte Identität ist ein Benutzer aus Ihrem Unternehmensverzeichnis, Ihrem Directory Service Web-Identitätsanbieter oder der AWS-Services mithilfe von Anmeldeinformationen

aus einer Identitätsquelle zugreift. Verbundene Identitäten übernehmen Rollen, die temporäre Anmeldeinformationen bereitstellen.

Für die zentrale Zugriffsverwaltung empfehlen wir AWS IAM Identity Center. Weitere Informationen finden Sie unter [Was ist IAM Identity Center?](#) im AWS IAM Identity Center -Benutzerhandbuch.

IAM-Benutzer und -Gruppen

Ein [IAM-Benutzer](#) ist eine Identität mit bestimmten Berechtigungen für eine einzelne Person oder Anwendung. Wir empfehlen die Verwendung temporärer Anmeldeinformationen anstelle von IAM-Benutzern mit langfristigen Anmeldeinformationen. Weitere Informationen finden Sie im IAM-Benutzerhandbuch unter [Erfordern, dass menschliche Benutzer den Verbund mit einem Identitätsanbieter verwenden müssen, um AWS mithilfe temporärer Anmeldeinformationen darauf zugreifen zu können](#).

Eine [IAM-Gruppe](#) spezifiziert eine Sammlung von IAM-Benutzern und erleichtert die Verwaltung von Berechtigungen für große Gruppen von Benutzern. Weitere Informationen finden Sie unter [Anwendungsfälle für IAM-Benutzer](#) im IAM-Benutzerhandbuch.

IAM-Rollen

Eine [IAM-Rolle](#) ist eine Identität mit spezifischen Berechtigungen, die temporäre Anmeldeinformationen bereitstellt. Sie können eine Rolle übernehmen, indem Sie [von einer Benutzer- zu einer IAM-Rolle \(Konsole\) wechseln](#) oder indem Sie eine AWS Oder-API-Operation AWS CLI aufrufen. Weitere Informationen finden Sie unter [Methoden, um eine Rolle zu übernehmen](#) im IAM-Benutzerhandbuch.

IAM-Rollen sind nützlich für den Verbundbenutzer-Zugriff, temporäre IAM-Benutzerberechtigungen, kontoübergreifenden Zugriff, serviceübergreifenden Zugriff und Anwendungen, die auf Amazon EC2 laufen. Weitere Informationen finden Sie unter [Kontoübergreifender Ressourcenzugriff in IAM](#) im IAM-Benutzerhandbuch.

Verwalten des Zugriffs mit Richtlinien

Sie kontrollieren den Zugriff, AWS indem Sie Richtlinien erstellen und diese an AWS Identitäten oder Ressourcen anhängen. Eine Richtlinie definiert Berechtigungen, wenn sie mit einer Identität oder Ressource verknüpft sind. AWS bewertet diese Richtlinien, wenn ein Principal eine Anfrage stellt. Die meisten Richtlinien werden AWS als JSON-Dokumente gespeichert. Weitere Informationen zu JSON-Richtliniendokumenten finden Sie unter [Übersicht über JSON-Richtlinien](#) im IAM-Benutzerhandbuch.

Mit Hilfe von Richtlinien legen Administratoren fest, wer Zugriff auf was hat, indem sie definieren, welches Prinzipal welche Aktionen auf welchen Ressourcen und unter welchen Bedingungen durchführen darf.

Standardmäßig haben Benutzer, Gruppen und Rollen keine Berechtigungen. Ein IAM-Administrator erstellt IAM-Richtlinien und fügt sie zu Rollen hinzu, die die Benutzer dann übernehmen können. IAM-Richtlinien definieren Berechtigungen unabhängig von der Methode, die zur Ausführung der Operation verwendet wird.

Identitätsbasierte Richtlinien

Identitätsbasierte Richtlinien sind JSON-Berechtigungsrichtliniendokumente, die Sie einer Identität (Benutzer, Gruppe oder Rolle) anfügen können. Diese Richtlinien steuern, welche Aktionen Identitäten für welche Ressourcen und unter welchen Bedingungen ausführen können. Informationen zum Erstellen identitätsbasierter Richtlinien finden Sie unter [Definieren benutzerdefinierter IAM-Berechtigungen mit vom Kunden verwalteten Richtlinien](#) im IAM-Benutzerhandbuch.

Identitätsbasierte Richtlinien können Inline-Richtlinien (direkt in eine einzelne Identität eingebettet) oder verwaltete Richtlinien (eigenständige Richtlinien, die mit mehreren Identitäten verbunden sind) sein. Informationen dazu, wie Sie zwischen verwalteten und Inline-Richtlinien wählen, finden Sie unter [Choose between managed policies and inline policies](#) im IAM-Benutzerhandbuch.

Ressourcenbasierte Richtlinien

Ressourcenbasierte Richtlinien sind JSON-Richtliniendokumente, die Sie an eine Ressource anfügen. Beispiele hierfür sind Vertrauensrichtlinien für IAM-Rollen und Amazon S3-Bucket-Richtlinien. In Services, die ressourcenbasierte Richtlinien unterstützen, können Service-Administratoren sie verwenden, um den Zugriff auf eine bestimmte Ressource zu steuern. Sie müssen in einer ressourcenbasierten Richtlinie [einen Prinzipal angeben](#).

Ressourcenbasierte Richtlinien sind Richtlinien innerhalb dieses Diensts. Sie können AWS verwaltete Richtlinien von IAM nicht in einer ressourcenbasierten Richtlinie verwenden.

Zugriffskontrolllisten () ACLs

Zugriffskontrolllisten (ACLs) steuern, welche Principals (Kontomitglieder, Benutzer oder Rollen) über Zugriffsberechtigungen für eine Ressource verfügen. ACLs ähneln ressourcenbasierten Richtlinien, verwenden jedoch nicht das JSON-Richtliniendokumentformat.

Amazon S3 und Amazon VPC sind Beispiele für Dienste, die Unterstützung ACLs bieten. AWS WAF
Weitere Informationen finden Sie unter [Übersicht über ACLs die Zugriffskontrollliste \(ACL\)](#) im Amazon Simple Storage Service Developer Guide.

Weitere Richtlinientypen

AWS unterstützt zusätzliche Richtlinientypen, mit denen die maximalen Berechtigungen festgelegt werden können, die durch gängigere Richtlinientypen gewährt werden:

- **Berechtigungsgrenzen** – Eine Berechtigungsgrenze legt die maximalen Berechtigungen fest, die eine identitätsbasierte Richtlinie einer IAM-Entität erteilen kann. Weitere Informationen finden Sie unter [Berechtigungsgrenzen für IAM-Entitäten](#) im -IAM-Benutzerhandbuch.
- **Richtlinien zur Dienstkontrolle (SCPs)** — Geben Sie die maximalen Berechtigungen für eine Organisation oder Organisationseinheit in an AWS Organizations. Weitere Informationen finden Sie unter [Service-Kontrollrichtlinien](#) im AWS Organizations -Benutzerhandbuch.
- **Richtlinien zur Ressourcenkontrolle (RCPs)** — Legen Sie die maximal verfügbaren Berechtigungen für Ressourcen in Ihren Konten fest. Weitere Informationen finden Sie im AWS Organizations Benutzerhandbuch unter [Richtlinien zur Ressourcenkontrolle \(RCPs\)](#).
- **Sitzungsrichtlinien** – Sitzungsrichtlinien sind erweiterte Richtlinien, die als Parameter übergeben werden, wenn Sie eine temporäre Sitzung für eine Rolle oder einen Verbundbenutzer erstellen. Weitere Informationen finden Sie unter [Sitzungsrichtlinien](#) im IAM-Benutzerhandbuch.

Mehrere Richtlinientypen

Wenn für eine Anfrage mehrere Arten von Richtlinien gelten, sind die daraus resultierenden Berechtigungen schwieriger zu verstehen. Informationen darüber, wie AWS bestimmt wird, ob eine Anfrage zulässig ist, wenn mehrere Richtlinientypen betroffen sind, finden Sie unter [Bewertungslogik für Richtlinien](#) im IAM-Benutzerhandbuch.

Wie AWS-Services arbeiten Sie mit IAM

Einen allgemeinen Überblick darüber, wie die meisten IAM-Funktionen AWS-Services funktionieren, finden Sie im [AWS IAM-Benutzerhandbuch unter Dienste, die mit IAM funktionieren](#).

Informationen zur Verwendung bestimmter Dienste AWS-Service mit IAM finden Sie im Abschnitt Sicherheit im Benutzerhandbuch des jeweiligen Dienstes.

Fehlerbehebung bei AWS Identität und Zugriff

Verwenden Sie die folgenden Informationen, um häufig auftretende Probleme zu diagnostizieren und zu beheben, die bei der Arbeit mit AWS und IAM auftreten können.

Themen

- [Ich bin nicht berechtigt, eine Aktion durchzuführen in AWS](#)
- [Ich bin nicht berechtigt, iam auszuführen: PassRole](#)
- [Ich möchte Personen außerhalb von mir den Zugriff AWS-Konto auf meine AWS Ressourcen ermöglichen](#)

Ich bin nicht berechtigt, eine Aktion durchzuführen in AWS

Wenn Sie eine Fehlermeldung erhalten, dass Sie nicht zur Durchführung einer Aktion berechtigt sind, müssen Ihre Richtlinien aktualisiert werden, damit Sie die Aktion durchführen können.

Der folgende Beispielfehler tritt auf, wenn der IAM-Benutzer `mateojackson` versucht, über die Konsole Details zu einer fiktiven *my-example-widget*-Ressource anzuzeigen, jedoch nicht über `aws:GetWidget`-Berechtigungen verfügt.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
aws:GetWidget on resource: my-example-widget
```

In diesem Fall muss die Richtlinie für den Benutzer `mateojackson` aktualisiert werden, damit er mit der `aws:GetWidget`-Aktion auf die *my-example-widget*-Ressource zugreifen kann.

Wenn Sie Hilfe benötigen, wenden Sie sich an Ihren AWS Administrator. Ihr Administrator hat Ihnen Ihre Anmeldeinformationen zur Verfügung gestellt.

Ich bin nicht berechtigt, iam auszuführen: PassRole

Wenn Sie die Fehlermeldung erhalten, dass Sie nicht zum Durchführen der `iam:PassRole`-Aktion autorisiert sind, müssen Ihre Richtlinien aktualisiert werden, um eine Rolle an AWS übergeben zu können.

Einige AWS-Services ermöglichen es Ihnen, eine bestehende Rolle an diesen Dienst zu übergeben, anstatt eine neue Servic Rolle oder eine dienstverknüpfte Rolle zu erstellen. Hierzu benötigen Sie Berechtigungen für die Übergabe der Rolle an den Dienst.

Der folgende Beispielfehler tritt auf, wenn ein IAM-Benutzer mit dem Namen `marymajor` versucht, die Konsole zu verwenden, um eine Aktion in AWS auszuführen. Die Aktion erfordert jedoch, dass der Service über Berechtigungen verfügt, die durch eine Servicerolle gewährt werden. Mary besitzt keine Berechtigungen für die Übergabe der Rolle an den Dienst.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
iam:PassRole
```

In diesem Fall müssen die Richtlinien von Mary aktualisiert werden, um die Aktion `iam:PassRole` ausführen zu können.

Wenn Sie Hilfe benötigen, wenden Sie sich an Ihren AWS Administrator. Ihr Administrator hat Ihnen Ihre Anmeldeinformationen zur Verfügung gestellt.

Ich möchte Personen außerhalb von mir den Zugriff AWS-Konto auf meine AWS Ressourcen ermöglichen

Sie können eine Rolle erstellen, mit der Benutzer in anderen Konten oder Personen außerhalb Ihrer Organisation auf Ihre Ressourcen zugreifen können. Sie können festlegen, wem die Übernahme der Rolle anvertraut wird. Für Dienste, die ressourcenbasierte Richtlinien oder Zugriffskontrolllisten (ACLs) unterstützen, können Sie diese Richtlinien verwenden, um Personen Zugriff auf Ihre Ressourcen zu gewähren.

Weitere Informationen dazu finden Sie hier:

- Informationen darüber, ob diese Funktionen AWS unterstützt werden, finden Sie unter [Wie AWS-Services arbeiten Sie mit IAM](#)
- Informationen dazu, wie Sie Zugriff auf Ihre Ressourcen gewähren können, AWS-Konten die Ihnen gehören, finden Sie im IAM-Benutzerhandbuch unter [Gewähren des Zugriffs auf einen IAM-Benutzer in einem anderen AWS-Konto, den Sie besitzen](#).
- Informationen dazu, wie Sie Dritten Zugriff auf Ihre Ressourcen gewähren können AWS-Konten, finden Sie [AWS-Konten im IAM-Benutzerhandbuch unter Gewähren des Zugriffs für Dritte](#).
- Informationen dazu, wie Sie über einen Identitätsverbund Zugriff gewähren, finden Sie unter [Gewähren von Zugriff für extern authentifizierte Benutzer \(Identitätsverbund\)](#) im IAM-Benutzerhandbuch.
- Informationen zum Unterschied zwischen der Verwendung von Rollen und ressourcenbasierten Richtlinien für den kontoübergreifenden Zugriff finden Sie unter [Kontoübergreifender Ressourcenzugriff in IAM](#) im IAM-Benutzerhandbuch.

Überprüfung der Einhaltung der Vorschriften für dieses AWS Produkt oder diese Dienstleistung

Informationen darüber, ob AWS-Service ein [AWS-Services in den Geltungsbereich bestimmter Compliance-Programme fällt, finden Sie unter Umfang nach Compliance-Programm AWS-Services unter](#) . Wählen Sie dort das Compliance-Programm aus, an dem Sie interessiert sind. Allgemeine Informationen finden Sie unter [AWS Compliance-Programme AWS](#) .

Sie können Prüfberichte von Drittanbietern unter herunterladen AWS Artifact. Weitere Informationen finden Sie unter [Berichte herunterladen unter](#) .

Ihre Verantwortung für die Einhaltung der Vorschriften bei der Nutzung AWS-Services hängt von der Vertraulichkeit Ihrer Daten, den Compliance-Zielen Ihres Unternehmens und den geltenden Gesetzen und Vorschriften ab. Weitere Informationen zu Ihrer Verantwortung für die Einhaltung der Vorschriften bei der Nutzung AWS-Services finden Sie in der [AWS Sicherheitsdokumentation](#).

Dieses AWS Produkt oder dieser Service folgt dem [Modell der gemeinsamen Verantwortung](#) in Bezug auf die spezifischen Amazon Web Services (AWS) -Services, die es unterstützt. Informationen zur AWS Servicesicherheit finden Sie auf der [Seite mit der Dokumentation zur AWS Servicesicherheit](#) und den [AWS Services, für die das AWS Compliance-Programm zur Einhaltung der](#) Vorschriften zuständig ist.

Ausfallsicherheit für dieses AWS Produkt oder diese Dienstleistung

Die AWS globale Infrastruktur basiert auf AWS-Regionen Availability Zones.

AWS-Regionen bieten mehrere physisch getrennte und isolierte Availability Zones, die über Netzwerke mit niedriger Latenz, hohem Durchsatz und hoher Redundanz miteinander verbunden sind.

Mithilfe von Availability Zones können Sie Anwendungen und Datenbanken erstellen und ausführen, die automatisch Failover zwischen Zonen ausführen, ohne dass es zu Unterbrechungen kommt. Availability Zones sind besser verfügbar, fehlertoleranter und skalierbarer als herkömmliche Infrastrukturen mit einem oder mehreren Rechenzentren.

Weitere Informationen zu AWS Regionen und Availability Zones finden Sie unter [AWS Globale Infrastruktur](#).

Dieses AWS Produkt oder dieser Service folgt dem [Modell der gemeinsamen Verantwortung](#) in Bezug auf die spezifischen Amazon Web Services (AWS) -Services, die es unterstützt. Informationen zur AWS Servicesicherheit finden Sie auf der [Seite mit der Dokumentation zur AWS Servicesicherheit](#) und den [AWS Services, für die das AWS Compliance-Programm zur Einhaltung der](#) Vorschriften zuständig ist.

Sicherheit der Infrastruktur für dieses AWS Produkt oder diesen Service

Dieses AWS Produkt oder dieser Dienst verwendet Managed Services und ist daher durch die AWS globale Netzwerksicherheit geschützt. Informationen zu AWS Sicherheitsdiensten und zum AWS Schutz der Infrastruktur finden Sie unter [AWS Cloud-Sicherheit](#). Informationen zum Entwerfen Ihrer AWS Umgebung unter Verwendung der bewährten Methoden für die Infrastruktursicherheit finden Sie unter [Infrastructure Protection](#) in Security Pillar AWS Well-Architected Framework.

Sie verwenden AWS veröffentlichte API-Aufrufe, um über das Netzwerk auf dieses AWS Produkt oder diesen Service zuzugreifen. Kunden müssen Folgendes unterstützen:

- Transport Layer Security (TLS). Wir benötigen TLS 1.2 und empfehlen TLS 1.3.
- Verschlüsselungs-Suiten mit Perfect Forward Secrecy (PFS) wie DHE (Ephemeral Diffie-Hellman) oder ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Die meisten modernen Systeme wie Java 7 und höher unterstützen diese Modi.

Außerdem müssen Anforderungen mit einer Zugriffsschlüssel-ID und einem geheimen Zugriffsschlüssel signiert sein, der einem IAM-Prinzipal zugeordnet ist. Alternativ können Sie mit [AWS -Security-Token-Service](#) (AWS STS) temporäre Sicherheitsanmeldeinformationen erstellen, um die Anforderungen zu signieren.

Dieses AWS Produkt oder dieser Service folgt dem [Modell der gemeinsamen Verantwortung](#) in Bezug auf die spezifischen Amazon Web Services (AWS) -Services, die es unterstützt. Informationen zur AWS Servicesicherheit finden Sie auf der [Seite mit der Dokumentation zur AWS Servicesicherheit](#) und den [AWS Services, für die das AWS Compliance-Programm zur Einhaltung der](#) Vorschriften zuständig ist.

Erzwingen Sie eine TLS-Mindestversion

Um die Sicherheit bei der Kommunikation mit AWS Diensten zu erhöhen, konfigurieren Sie den AWS SDK für JavaScript für die Verwendung von TLS 1.2 oder höher.

Transport Layer Security (TLS) ist ein Protokoll, das von Webbrowsern und anderen Anwendungen verwendet wird, um die Privatsphäre und Integrität der über ein Netzwerk ausgetauschten Daten zu gewährleisten.

Important

Am 10. Juni 2024 haben wir [angekündigt](#), dass TLS 1.3 auf AWS Service-API-Endpunkten in allen AWS Regionen verfügbar ist. Die Version AWS SDK für JavaScript v3 handelt nicht selbst über die TLS-Version aus. Stattdessen wird die von Node.js festgelegte TLS-Version verwendet, die über konfiguriert werden kann `https.Agent`. AWS empfiehlt die Verwendung der aktuellen Active LTS-Version von Node.js.

Überprüfen und Erzwingen von TLS in Node.js

Wenn Sie die AWS SDK für JavaScript zusammen mit Node.js verwenden, wird die zugrunde liegende Sicherheitsebene Node.js verwendet, um die TLS-Version festzulegen.

Node.js 12.0.0 und höher verwenden eine Mindestversion von OpenSSL 1.1.1b, die TLS 1.3 unterstützt. Node.js verwendet standardmäßig TLS 1.3, sofern verfügbar. Sie können bei Bedarf explizit eine andere Version angeben.

Überprüfen der Version von OpenSSL und TLS

Führen Sie den folgenden Befehl aus, um die von Node.js verwendete Version von OpenSSL auf Ihrem Computer abzurufen.

```
node -p process.versions
```

Die Version von OpenSSL in der Liste ist die von Node.js verwendete Version, wie im folgenden Beispiel gezeigt.

```
openssl: '1.1.1b'
```

Um die von Node.js verwendete Version von TLS auf Ihrem Computer abzurufen, starten Sie die Knoten-Shell und führen Sie die folgenden Befehle in der angegebenen Reihenfolge aus.

```
> var tls = require("tls");  
> var tlsSocket = new tls.TLSSocket();  
> tlsSocket.getProtocol();
```

Der letzte Befehl gibt die TLS-Version aus, wie im folgenden Beispiel gezeigt.

```
'TLSv1.3'
```

Node.js verwendet standardmäßig diese Version von TLS und versucht, eine andere Version von TLS auszuhandeln, wenn ein Aufruf nicht erfolgreich ist.

Überprüfung der unterstützten Mindest- und Höchstwerte der unterstützten TLS-Versionen

Entwickler können die Mindest- und Höchstzahl der unterstützten TLS-Versionen in Node.js mithilfe des folgenden Skripts überprüfen:

```
import tls from "tls";  
console.log("Supported TLS versions:", tls.DEFAULT_MIN_VERSION + " to " +  
  tls.DEFAULT_MAX_VERSION);
```

Der letzte Befehl gibt die standardmäßige minimale und maximale TLS-Version aus, wie im folgenden Beispiel gezeigt.

```
Supported TLS versions: TLSv1.2 to TLSv1.3
```

Erzwingen einer Mindestversion von TLS

Node.js verhandelt eine Version von TLS, wenn ein Aufruf fehlschlägt. Sie können die minimal zulässige TLS-Version während dieser Verhandlung erzwingen, entweder wenn Sie ein Skript über die Befehlszeile ausführen oder pro Anfrage in Ihrem JavaScript Code.

Um die TLS-Mindestversion von der Befehlszeile aus anzugeben, müssen Sie Node.js Version 11.4.0 oder höher verwenden. Um eine bestimmte Version von Node.js zu installieren, installieren Sie zunächst Node Version Manager (nvm) mithilfe der unter [Installation und Aktualisierung des Node Version Managers beschrieben](#) Schritte. Führen Sie dann die folgenden Befehle aus, um eine bestimmte Version von Node.js zu installieren und zu verwenden.

```
nvm install 11  
nvm use 11
```

Enforce TLS 1.2

Um zu erzwingen, dass TLS 1.2 die minimal zulässige Version ist, geben Sie beim Ausführen des Skripts das Argument `--tls-min-v1.2` an, wie im folgenden Beispiel gezeigt.

```
node --tls-min-v1.2 yourScript.js
```

Um die zulässige TLS-Mindestversion für eine bestimmte Anfrage in Ihrem JavaScript Code anzugeben, verwenden Sie den `minVersion` Parameter, um das Protokoll anzugeben, wie im folgenden Beispiel gezeigt.

```
import https from "https";  
import { NodeHttpHandler } from "@smithy/node-http-handler";  
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";  
  
const client = new DynamoDBClient({  
  region: "us-west-2",  
  requestHandler: new NodeHttpHandler({  
    httpsAgent: new https.Agent(  
      {  
        minVersion: 'TLSv1.2'  
      }  
    )  
  })  
});
```

Enforce TLS 1.3

Um zu erzwingen, dass TLS 1.3 die zulässige Mindestversion ist, geben Sie das `--tls-min-v1.3` Argument bei der Ausführung Ihres Skripts an, wie im folgenden Beispiel gezeigt.

```
node --tls-min-v1.3 yourScript.js
```

Um die zulässige TLS-Mindestversion für eine bestimmte Anfrage in Ihrem JavaScript Code anzugeben, verwenden Sie den `minVersion` Parameter, um das Protokoll anzugeben, wie im folgenden Beispiel gezeigt.

```
import https from "https";
import { NodeHttpHandler } from "@smithy/node-http-handler";
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";

const client = new DynamoDBClient({
  region: "us-west-2",
  requestHandler: new NodeHttpHandler({
    httpsAgent: new https.Agent({
      {
        minVersion: 'TLSv1.3'
      }
    })
  })
});
```

Überprüfen und Erzwingen von TLS in einem Browserskript

Wenn Sie das SDK für JavaScript in einem Browserskript verwenden, steuern die Browsereinstellungen die verwendete TLS-Version. Die vom Browser verwendete Version von TLS kann nicht vom Skript erkannt oder festgelegt werden und muss vom Benutzer konfiguriert werden. Informationen zur Überprüfung und Erzwingung der in einem Browserskript verwendeten Version von TLS finden Sie in den Anweisungen für den jeweiligen Browser.

Microsoft Internet Explorer

1. Öffnen Sie Internet Explorer.
2. Wählen Sie in der Menüleiste die Option Extras — Internetoptionen — Registerkarte Erweitert.
3. Scrollen Sie nach unten zur Kategorie Sicherheit und aktivieren Sie manuell das Optionsfeld für TLS 1.2 verwenden.
4. Klicken Sie auf OK.
5. Schließen Sie Ihren Browser und starten Sie Internet Explorer neu.

Microsoft Edge

1. Geben Sie im Suchfeld des Windows-Menüs Folgendes ein *Internet options*.
2. Klicken Sie unter Beste Übereinstimmung auf Internetoptionen.

3. Scrollen Sie im Fenster Interneteneigenschaften auf der Registerkarte Erweitert nach unten zum Abschnitt Sicherheit.
4. Markieren Sie das Kontrollkästchen User TLS 1.2.
5. Klicken Sie auf OK.

Google Chrome

1. Öffnen Sie Google Chrome.
2. Klicken Sie auf Alt F und wählen Sie Einstellungen.
3. Scrollen Sie nach unten und wählen Sie Erweiterte Einstellungen anzeigen... .
4. Scrollen Sie nach unten zum Abschnitt System und klicken Sie auf Proxyeinstellungen öffnen... .
5. Wählen Sie die Registerkarte Erweitert aus.
6. Scrollen Sie nach unten zur Kategorie Sicherheit und aktivieren Sie manuell das Optionsfeld für TLS 1.2 verwenden.
7. Klicken Sie auf OK.
8. Schließen Sie Ihren Browser und starten Sie Google Chrome neu.

Mozilla Firefox

1. Öffnen Sie Firefox.
2. Geben Sie in der Adressleiste about:config ein und drücken Sie die Eingabetaste.
3. Geben Sie im Suchfeld tls ein. Suchen Sie den Eintrag für security.tls.version.min und doppelklicken Sie darauf.
4. Setzen Sie den Integer-Wert auf 3, um zu erzwingen, dass das Protokoll von TLS 1.2 als Standard verwendet wird.
5. Klicken Sie auf OK.
6. Schließen Sie Ihren Browser und starten Sie Mozilla Firefox neu.

Apple Safari

Es gibt keine Optionen zum Aktivieren von SSL-Protokollen. Wenn Sie Safari Version 7 oder höher verwenden, wird TLS 1.2 automatisch aktiviert.

Die TLS-Version wird in AWS SDK für JavaScript v3-Anfragen abgerufen

Sie können die in einer AWS SDK-Anfrage verwendete TLS-Version mit dem folgenden Skript protokollieren:

```
import { S3Client, ListBucketsCommand } from "@aws-sdk/client-s3";
import tls from "tls";

const client = new S3Client({ region: "us-east-1" });

const tlsSocket = new tls.TLSSocket();

client.middlewareStack.add((next, context) => async (args) => {
  console.log(`Using TLS version: ${tlsSocket.getProtocol()}`);
  return next(args);
});
```

Der letzte Befehl gibt die verwendete TLS-Version aus, wie im folgenden Beispiel gezeigt.

```
Using TLS version: TLSv1.3
```

Migrieren Sie von Version 2.x auf 3.x von AWS SDK für JavaScript

Die AWS SDK für JavaScript Version 3 ist eine umfassende Neufassung von Version 2. Der Abschnitt beschreibt die Unterschiede zwischen den beiden Versionen und erklärt, wie Sie von Version 2 auf Version 3 des SDK für JavaScript migrieren.

Migrieren Sie Ihren Code mithilfe von Codemod auf SDK für JavaScript v3

AWS SDK für JavaScript Version 3 (v3) bietet modernisierte Schnittstellen für Client-Konfigurationen und Dienstprogramme, zu denen Anmeldeinformationen, mehrteiliger Amazon S3 S3-Upload, DynamoDB-Dokumentencient, Kellner und mehr gehören. [Was sich in Version 2 geändert hat, und was in Version 3 für jede Änderung geändert wurde, finden Sie im Migrationsleitfaden im Repo. AWS SDK für JavaScript GitHub](#)

Um die Vorteile der Version 3 voll auszuschöpfen, empfehlen AWS SDK für JavaScript wir die Verwendung der unten beschriebenen Codemod-Skripte.

Verwenden Sie Codemod, um vorhandenen v2-Code zu migrieren

Die Sammlung von Codemod-Skripten in [aws-sdk-js-codemod](#) hilft Ihnen bei der Migration Ihrer vorhandenen AWS SDK für JavaScript (v2) -Anwendung zur Verwendung von v3. APIs Sie können die Transformation wie folgt ausführen.

```
$ npx aws-sdk-js-codemod -t v2-to-v3 PATH...
```

Stellen Sie sich zum Beispiel vor, Sie haben den folgenden Code, der einen Amazon DynamoDB-Client aus Version 2 erstellt und Operation aufruft `listTables`.

```
// example.ts
import AWS from "aws-sdk";

const region = "us-west-2";
const client = new AWS.DynamoDB({ region });
await client.listTables({}).promise()
```

```
.then(console.log)
.catch(console.error);
```

Sie können unsere v2-to-v3 Transformation `example.ts` wie folgt ausführen.

```
$ npx aws-sdk-js-codemod -t v2-to-v3 example.ts
```

Die Transformation konvertiert den DynamoDB-Import in Version 3, erstellt einen v3-Client und ruft den `listTables` Vorgang wie folgt auf.

```
// example.ts
import { DynamoDB } from "@aws-sdk/client-dynamodb";

const region = "us-west-2";
const client = new DynamoDB({ region });
await client.listTables({})
  .then(console.log)
  .catch(console.error);
```

Wir haben Transformationen für gängige Anwendungsfälle implementiert. Wenn Ihr Code nicht korrekt transformiert wird, erstellen Sie bitte einen [Fehlerbericht](#) oder eine [Funktionsanfrage](#) mit Beispieleingabe- und observed/expected Ausgabecode. Wenn Ihr spezieller Anwendungsfall bereits in [einem bestehenden Problem gemeldet wurde](#), zeigen Sie Ihre Unterstützung durch eine positive Bewertung.

Was ist neu in Version 3

Version 3 des SDK für JavaScript (v3) enthält die folgenden neuen Funktionen.

Modularisierte Pakete

Benutzer können jetzt für jeden Dienst ein separates Paket verwenden.

Neuer Middleware-Stack

Benutzer können jetzt einen Middleware-Stack verwenden, um den Lebenszyklus eines Operationsaufrufs zu steuern.

Darüber hinaus ist das SDK integriert, was viele Vorteile bietet TypeScript, wie z. B. die statische Typisierung.

 Important

Die Codebeispiele für Version 3 in diesem Handbuch sind in ECMAScript 6 (ES6) geschrieben. ES6 bietet neue Syntax und neue Funktionen, um Ihren Code moderner und lesbarer zu machen und mehr zu erreichen. ES6 erfordert, dass Sie Node.js Version 13.x oder höher verwenden. Informationen zum Herunterladen und Installieren der neuesten Version von Node.js finden Sie unter [Node.js downloads](#). Weitere Informationen finden Sie unter [JavaScript ES6/CommonJs-Syntax](#).

Modularisierte Pakete

Für Version 2 des SDK für JavaScript (v2) mussten Sie das gesamte AWS SDK wie folgt verwenden.

```
var AWS = require("aws-sdk");
```

Das Laden des gesamten SDK ist kein Problem, wenn Ihre Anwendung viele AWS Dienste verwendet. Wenn Sie jedoch nur wenige AWS Dienste verwenden müssen, bedeutet dies, dass Sie die Größe Ihrer Anwendung mit Code erhöhen müssen, den Sie nicht benötigen oder verwenden.

In Version 3 können Sie nur die einzelnen AWS Dienste laden und verwenden, die Sie benötigen. Dies wird im folgenden Beispiel gezeigt, das Ihnen Zugriff auf Amazon DynamoDB (DynamoDB) gewährt.

```
import { DynamoDB } from "@aws-sdk/client-dynamodb";
```

Sie können nicht nur einzelne AWS Dienste laden und verwenden, sondern Sie können auch nur die Servicebefehle laden und verwenden, die Sie benötigen. Dies wird in den folgenden Beispielen veranschaulicht, die Ihnen Zugriff auf den DynamoDB-Client und den `ListTablesCommand` Befehl geben.

```
import {  
  DynamoDBClient,  
  ListTablesCommand  
} from "@aws-sdk/client-dynamodb";
```

Important

Sie sollten Submodule nicht in Module importieren. Beispielsweise kann der folgende Code zu Fehlern führen.

```
import { CognitoIdentity } from "@aws-sdk/client-cognito-identity/  
CognitoIdentity";
```

Der folgende Code ist korrekt.

```
import { CognitoIdentity } from "@aws-sdk/client-cognito-identity";
```

Codegröße vergleichen

In Version 2 (v2) könnte ein einfaches Codebeispiel, das alle Ihre Amazon DynamoDB-Tabellen in der us-west-2 Region auflistet, wie folgt aussehen.

```
var AWS = require("aws-sdk");  
// Set the Region  
AWS.config.update({ region: "us-west-2" });  
// Create DynamoDB service object  
var ddb = new AWS.DynamoDB({ apiVersion: "2012-08-10" });  
  
// Call DynamoDB to retrieve the list of tables  
ddb.listTables({ Limit: 10 }, function (err, data) {  
  if (err) {  
    console.log("Error", err.code);  
  } else {  
    console.log("Tables names are ", data.TableNames);  
  }  
});
```

v3 sieht wie folgt aus.

```
import {  
  DynamoDBClient,  
  ListTablesCommand  
} from "@aws-sdk/client-dynamodb";
```

```
const dbclient = new DynamoDBClient({ region: "us-west-2" });

try {
  const results = await dbclient.send(new ListTablesCommand);

  for (const item of results.TableNames) {
    console.log(item);
  }
} catch (err) {
  console.error(err)
}
```

Das `aws-sdk` Paket erweitert Ihre Anwendung um etwa 40 MB. Durch das `var AWS = require("aws-sdk")` Ersetzen durch `import {DynamoDB} from "@aws-sdk/client-dynamodb"` wird dieser Overhead auf etwa 3 MB reduziert. Durch die Beschränkung des Imports auf den DynamoDB-Client und den `ListTablesCommand` Befehl wird der Overhead auf weniger als 100 KB reduziert.

```
// Load the DynamoDB client and ListTablesCommand command for Node.js
import {
  DynamoDBClient,
  ListTablesCommand
} from "@aws-sdk/client-dynamodb";
const dbclient = new DynamoDBClient({});
```

Befehle in Version 3 aufrufen

Sie können Operationen in Version 3 entweder mit v2- oder v3-Befehlen ausführen. Um v3-Befehle zu verwenden, importieren Sie die Befehle und die erforderlichen AWS Services-Paketclients und führen den Befehl mithilfe der `.send` Methode aus, die das `async/await` Muster verwendet.

Um v2-Befehle zu verwenden, importieren Sie die erforderlichen AWS Services-Pakete und führen den v2-Befehl mithilfe eines Callbacks oder eines `async/await` Musters direkt im Paket aus.

Verwenden von v3-Befehlen

v3 stellt für jedes AWS Servicepaket eine Reihe von Befehlen bereit, mit denen Sie Operationen für diesen AWS Dienst ausführen können. Nachdem Sie einen AWS Dienst installiert haben, können Sie die verfügbaren Befehle in Ihrem Projekt durchsuchen `node_modules/@aws-sdk/client-PACKAGE_NAME/commands` folder.

Sie müssen die Befehle importieren, die Sie verwenden möchten. Der folgende Code lädt beispielsweise den DynamoDB-Dienst und den CreateTableCommand Befehl.

```
import { DynamoDB, CreateTableCommand } from "@aws-sdk/client-dynamodb";
```

Verwenden Sie die folgende Syntax, um diese Befehle im empfohlenen async/await Muster aufzurufen.

```
CLIENT.send(new XXXCommand);
```

Im folgenden Beispiel wird beispielsweise eine DynamoDB-Tabelle mit dem empfohlenen async/await Muster erstellt.

```
import { DynamoDB, CreateTableCommand } from "@aws-sdk/client-dynamodb";
const dynamodb = new DynamoDB({ region: "us-west-2" });
const tableParams = {
  TableName: TABLE_NAME
};

try {
  const data = await dynamodb.send(new CreateTableCommand(tableParams));
  console.log("Success", data);
} catch (err) {
  console.log("Error", err);
};
```

Verwenden von v2-Befehlen

Um v2-Befehle im SDK für zu verwenden JavaScript, importieren Sie die vollständigen AWS Servicepakete, wie im folgenden Code gezeigt.

```
const { DynamoDB } = require('@aws-sdk/client-dynamodb');
```

Verwenden Sie die folgende Syntax, um v2-Befehle im empfohlenen async/await Muster aufzurufen.

```
client.command(parameters);
```

Im folgenden Beispiel wird der createTable v2-Befehl verwendet, um eine DynamoDB-Tabelle mit dem empfohlenen async/await Muster zu erstellen.

```
const { DynamoDB } = require('@aws-sdk/client-dynamodb');
const dynamoDB = new DynamoDB({ region: 'us-west-2' });
var tableParams = {
  TableName: TABLE_NAME
};
async function run() => {
  try {
    const data = await dynamoDB.createTable(tableParams);
    console.log("Success", data);
  }
  catch (err) {
    console.log("Error", err);
  }
};
run();
```

Im folgenden Beispiel wird der `createBucket v2`-Befehl verwendet, um einen Amazon S3 S3-Bucket mithilfe des Callback-Musters zu erstellen.

```
const { S3 } = require('@aws-sdk/client-s3');
const s3 = new S3({ region: 'us-west-2' });
var bucketParams = {
  Bucket : BUCKET_NAME
};
function run() {
  s3.createBucket(bucketParams, function (err, data) {
    if (err) {
      console.log("Error", err);
    } else {
      console.log("Success", data.Location);
    }
  })
};
run();
```

Neuer Middleware-Stack

Version 2 des SDK ermöglichte es Ihnen, eine Anfrage in den verschiedenen Phasen ihres Lebenszyklus zu ändern, indem Sie der Anfrage Event-Listener anhängen. Dieser Ansatz kann es schwierig machen, Fehler zu debuggen, die während des Lebenszyklus einer Anfrage schief gelaufen sind.

In Version 3 können Sie einen neuen Middleware-Stack verwenden, um den Lebenszyklus eines Operationsaufrufs zu steuern. Dieser Ansatz bietet eine Reihe von Vorteilen. Jede Middleware-Phase im Stack ruft die nächste Middleware-Stufe auf, nachdem Änderungen am Anforderungsobjekt vorgenommen wurden. Dies erleichtert auch das Debuggen von Problemen im Stack erheblich, da Sie genau sehen können, welche Middleware-Stufen aufgerufen wurden, bevor der Fehler aufgetreten ist.

Das folgende Beispiel fügt einem Amazon DynamoDB-Client (den wir zuvor erstellt und gezeigt haben) mithilfe von Middleware einen benutzerdefinierten Header hinzu. Das erste Argument ist eine Funktion, die akzeptiert `next`, was die nächste aufzurufende Middleware-Stufe im Stack ist, und ein `ObjectContext`, das einige Informationen über die aufgerufene Operation enthält. Die Funktion gibt eine Funktion zurück, die akzeptiert `args`, d. h. ein Objekt, das die an den Vorgang und die Anforderung übergebenen Parameter enthält. Sie gibt das Ergebnis des Aufrufs der nächsten Middleware mit zurück. `args`

```
dbclient.middlewareStack.add(  
  (next, context) => args => {  
    args.request.headers["Custom-Header"] = "value";  
    return next(args);  
  },  
  {  
    name: "my-middleware",  
    override: true,  
    step: "build"  
  }  
);  
  
dbclient.send(new PutObjectCommand(params));
```

Was ist der Unterschied zwischen AWS SDK für JavaScript v2 und v3?

In diesem Abschnitt werden die wichtigsten Änderungen von AWS SDK für JavaScript Version 2 zu Version 3 beschrieben. Da v3 eine modulare Neufassung von v2 ist, unterscheiden sich einige grundlegende Konzepte zwischen v2 und v3. In unseren [Blogbeiträgen](#) erfahren Sie mehr über diese Änderungen. Die folgenden Blogbeiträge werden Sie auf den neuesten Stand bringen:

- [Modulare Pakete in AWS SDK für JavaScript](#)
- [Wir stellen vor: Middleware Stack in Modular AWS SDK für JavaScript](#)

Die Zusammenfassung der Schnittstellenänderungen von AWS SDK für JavaScript v2 zu v3 ist unten angegeben. Ziel ist es, Ihnen zu helfen, die v3-Äquivalente der Version 2, mit der APIs Sie bereits vertraut sind, leicht zu finden.

Themen

- [Konstrukteure des Kunden](#)
- [Anbieter von Anmeldeinformationen](#)
- [Überlegungen zu Amazon S3](#)
- [DynamoDB-Dokumententclient](#)
- [Kellner und Unterzeichner](#)
- [Hinweise zu bestimmten Servicekunden](#)

Konstrukteure des Kunden

Diese Liste ist nach [v2-Konfigurationsparametern](#) indexiert.

- [computeChecksums](#)
 - v2: Ob MD5 Prüfsummen für Nutzlastkörper berechnet werden sollen, wenn der Dienst sie akzeptiert (derzeit nur in S3 unterstützt).
 - v3: Die entsprechenden Befehle von S3 (PutObject PutBucketCors, usw.) berechnen automatisch die MD5 Prüfsummen für die Nutzdaten der Anfrage. Sie können auch einen anderen Prüfsummenalgorithmus im ChecksumAlgorithm Parameter der Befehle angeben, um einen anderen Prüfsummenalgorithmus zu verwenden. Weitere Informationen finden Sie in der Ankündigung der [S3-Funktion](#).
- [convertResponseTypes](#)
 - v2: Ob Typen beim Parsen von Antwortdaten konvertiert werden.
 - v3: Veraltet. Diese Option gilt als nicht typsicher, da sie Typen wie Timestamp oder Base64-Binärdateien nicht aus der JSON-Antwort konvertiert.
- [correctClockSkew](#)
 - v2: Ob eine Korrektur der Zeitversetzung angewendet und Anfragen wiederholt werden sollen, die aufgrund einer schiefen Client-Uhr fehlschlagen.
 - v3: Veraltet. Das SDK wendet immer eine Korrektur der Taktverzerrung an.
- [systemClockOffset](#)
 - v2: Ein Offsetwert in Millisekunden, der für alle Signierzeiten gilt.
 - v3: Keine Änderung.

- [credentials](#)
 - v2: Die AWS Anmeldeinformationen, mit denen Anfragen signiert werden sollen.
 - v3: Keine Änderung. Es kann sich auch um eine asynchrone Funktion handeln, die Anmeldeinformationen zurückgibt. Wenn die Funktion ein `zurückgibtexpiration` (Date), wird die Funktion erneut aufgerufen, wenn sich das Ablaufdatum nähert. Anmeldeinformationen finden Sie in der [v3-API-Referenz. `AwsAuthInputConfig`](#)
- [endpointCacheSize](#)
 - v2: Die Größe des globalen Caches, in dem Endpunkte aus Endpoint Discovery-Vorgängen gespeichert werden.
 - v3: Keine Änderung.
- [endpointDiscoveryEnabled](#)
 - v2: Ob Operationen mit vom Dienst angegebenen Endpunkten dynamisch aufgerufen werden sollen.
 - v3: Keine Änderung.
- [hostPrefixEnabled](#)
 - v2: Ob Anforderungsparameter dem Präfix des Hostnamens zugewiesen werden sollen.
 - v3: Veraltet. Das SDK fügt bei Bedarf immer das Hostnamenpräfix ein.
- [httpOptions](#)

Eine Reihe von Optionen, die an die Low-Level-HTTP-Anfrage übergeben werden. Diese Optionen sind in Version 3 unterschiedlich aggregiert. Sie können sie konfigurieren, indem Sie eine neue `requestHandler` angeben. Hier ist das Beispiel für die Einstellung von HTTP-Optionen in der Laufzeit von Node.js. Weitere Informationen finden Sie in der [v3-API-Referenz für `NodeHttpHandler`](#).

Alle v3-Anfragen verwenden standardmäßig HTTPS. Sie müssen nur einen benutzerdefinierten `HTTPSAgent` bereitstellen.

```
const { Agent } = require("https");
const { Agent: HttpAgent } = require("http");
const { NodeHttpHandler } = require("@smithy/node-http-handler");
const dynamodbClient = new DynamoDBClient({
  requestHandler: new NodeHttpHandler({
    httpsAgent: new Agent({
      /*params*/
    }),
    connectionTimeout: /*number in milliseconds*/,
    socketTimeout: /*number in milliseconds*/
  })
});
```



```
    }},
  });
```

Wenn Sie einen benutzerdefinierten Endpunkt übergeben, der http verwendet, müssen Sie `HttpAgent` angeben.

```
const { Agent } = require("http");
const { NodeHttpHandler } = require("@smithy/node-http-handler");

const dynamodbClient = new DynamoDBClient({
  requestHandler: new NodeHttpHandler({
    httpAgent: new Agent({
      /*params*/
    }),
  }),
  endpoint: "http://example.com",
});
```

Wenn der Client in Browsern ausgeführt wird, ist ein anderer Satz von Optionen verfügbar. Weitere Informationen finden Sie in der [v3-API-Referenz für FetchHttpHandler](#).

```
const { FetchHttpHandler } = require("@smithy/fetch-http-handler");
const dynamodbClient = new DynamoDBClient({
  requestHandler: new FetchHttpHandler({
    requestTimeout: /* number in milliseconds */
  }),
});
```

Jede Option von `httpOptions` ist unten angegeben:

- `proxy`
 - v2: Die URL, über die Anfragen weitergeleitet werden sollen.
 - v3: Sie können im Folgenden [Proxys für Node.js konfigurieren einen Proxy](#) mit einem Agenten einrichten.
- `agent`
 - v2: Das Agent-Objekt, mit dem HTTP-Anfragen ausgeführt werden sollen. Wird für das Verbindungspooling verwendet.
 - v3: Sie können `httpAgent` oder `httpsAgent` wie in den obigen Beispielen gezeigt konfigurieren.

- `connectTimeout`

- v2: Setzt den Socket auf Timeout, wenn nach connectTimeout Millisekunden keine Verbindung zum Server hergestellt werden konnte.
- v3: [connectionTimeout ist in den Optionen verfügbar. NodeHttpHandler](#)
- timeout
 - v2: Die Anzahl der Millisekunden, die eine Anfrage dauern kann, bevor sie automatisch beendet wird.
 - v3: [socketTimeout ist in den Optionen verfügbar. NodeHttpHandler](#)
- xhrAsync
 - v2: Ob das SDK asynchrone HTTP-Anfragen sendet.
 - v3: Veraltet. Anfragen sind immer asynchron.
- xhrWithCredentials
 - v2: Legt die Eigenschaft „withCredentials“ eines XMLHttpRequest-Objekts fest.
 - v3: Nicht verfügbar. Das SDK erbt [die Standard-Abrufkonfigurationen](#).
- [logger](#)
 - v2: Ein Objekt, das auf .write() (wie ein Stream) oder .log() (wie das Konsolenobjekt) reagiert, um Informationen über Anfragen zu protokollieren.
 - v3: Keine Änderung. Detailliertere Protokolle sind in Version 3 verfügbar.
- [maxRedirects](#)
 - v2: Die maximale Anzahl an Weiterleitungen, denen bei einer Serviceanfrage gefolgt werden muss.
 - v3: Veraltet. Das SDK folgt keinen Weiterleitungen, um unbeabsichtigte regionsübergreifende Anfragen zu vermeiden.
- [maxRetries](#)
 - v2: Die maximale Anzahl an Wiederholungen, die für eine Serviceanfrage ausgeführt werden können.
 - v3: Geändert zu. maxAttempts Weitere Informationen finden Sie in der [v3-API-Referenz für RetryInputConfig](#). Beachten Sie, dass dies der maxAttempts Fall sein soll `maxRetries + 1`.
- [paramValidation](#)
 - v2: Ob Eingabeparameter vor dem Senden der Anfrage anhand der Vorgangsbeschreibung überprüft werden sollen.
 - v3: Veraltet. Das SDK führt zur Laufzeit keine Validierung auf der Clientseite durch.
- [region](#)
 - v2: Die Region, an die Serviceanfragen gesendet werden sollen.
 - v3: Keine Änderung. Es kann sich auch um eine asynchrone Funktion handeln, die eine Regionszeichenfolge zurückgibt.

- [retryDelayOptions](#)
 - v2: Eine Reihe von Optionen zur Konfiguration der Wiederholungsverzögerung bei wiederholten Fehlern.
 - v3: Veraltet. Das SDK unterstützt eine flexiblere Wiederholungsstrategie mit der `retryStrategy` Client-Konstruktor-Option. Weitere Informationen finden Sie [in der API-Referenz für Version 3](#).
- [s3BucketEndpoint](#)
 - v2: Ob der angegebene Endpunkt einen einzelnen Bucket adressiert (falsch, wenn er den Root-API-Endpunkt adressiert).
 - v3: Geändert zu `bucketEndpoint`. Weitere Informationen finden Sie in der [v3-API-Referenz für BucketEndpoint](#). Beachten Sie, dass `true`, wenn Sie den Anforderungsendpunkt im Anforderungsparameter `bucket` auf `true` angeben, der ursprüngliche Endpunkt überschrieben wird. In Version 2 hingegen überschreibt der Anforderungsendpunkt im Client-Konstruktor den Bucket Anforderungsparameter.
- [s3DisableBodySigning](#)
 - v2: Ob die S3-Bodysignatur deaktiviert werden soll, wenn die Signaturversion v4 verwendet wird.
 - v3: Umbenannt in `disableChecksum`.
- [s3ForcePathStyle](#)
 - v2: Ob der Pfadstil URLs für S3-Objekte erzwungen werden soll.
 - v3: Umbenannt in `forcePathStyle`.
- [s3UseArnRegion](#)
 - v2: Ob die Anforderungsregion mit der Region überschrieben werden soll, die aus dem ARN der angeforderten Ressource abgeleitet wird.
 - v3: Umbenannt in `useArnRegion`.
- [s3UseEast1RegionalEndpoint](#)
 - v2: Wenn die Region auf 'us-east-1' gesetzt ist, ob eine S3-Anfrage an globale Endpunkte oder an regionale Endpunkte 'us-east-1' gesendet werden soll.
 - v3: Veraltet. Der S3-Client verwendet immer den regionalen Endpunkt, wenn die Region auf eingestellt ist. `us-east-1` Sie können die Region so einstellen, dass Anfragen `aws-global` an den globalen S3-Endpunkt gesendet werden.
- [signatureCache](#)
 - v2: Ob die Signatur, mit der Anfragen signiert werden sollen (außer Kraft der API-Konfiguration), zwischengespeichert wird.
 - v3: Veraltet. Das SDK speichert die Hash-Signaturschlüssel immer zwischen.
- [signatureVersion](#)

- v2: Die Signaturversion, mit der Anfragen signiert werden sollen (überschreibt die API-Konfiguration).
- v3: Veraltet. Die im v2-SDK unterstützte Signatur V2 wurde von als veraltet eingestuft AWS. v3 unterstützt nur die Signatur v4.
- [sslEnabled](#)
 - v2: Ob SSL für Anfragen aktiviert ist.
 - v3: Umbenannt in `tls`.
- [stsRegionalEndpoints](#)
 - v2: Ob eine STS-Anfrage an globale oder regionale Endpunkte gesendet werden soll.
 - v3: Veraltet. Der STS-Client verwendet immer regionale Endpunkte, wenn er auf eine bestimmte Region eingestellt ist. Sie können die Region auf einstellen, um eine Anfrage `aws-global` an den globalen STS-Endpunkt zu senden.
- [useAccelerateEndpoint](#)
 - v2: Ob der Accelerate-Endpunkt mit dem S3-Dienst verwendet werden soll.
 - v3: Keine Änderung.

Anbieter von Anmeldeinformationen

In Version 2 JavaScript stellt das SDK für eine Liste von Anmeldeinformationsanbietern zur Auswahl sowie eine Anbieterkette für Anmeldeinformationen bereit, die standardmäßig auf Node.js verfügbar ist und versucht, die AWS Anmeldeinformationen von allen gängigen Anbietern zu laden. Das SDK für JavaScript Version 3 vereinfacht die Benutzeroberfläche des Anmeldeinformationsanbieters und macht es einfacher, benutzerdefinierte Anbieter für Anmeldeinformationen zu verwenden und zu schreiben. Zusätzlich zu einer neuen Anbieterkette für Anmeldeinformationen bietet das SDK für JavaScript Version 3 eine Liste von Anbietern von Anmeldeinformationen, die ein Äquivalent zu Version 2 anbieten sollen.

Hier sind alle Anbieter von Anmeldeinformationen in Version 2 und ihre Entsprechungen in Version 3.

Standardanbieter für Anmeldeinformationen

Der Standardanbieter für Anmeldeinformationen ist die Art und Weise, wie das SDK die AWS Anmeldeinformationen JavaScript auflöst, wenn Sie keinen explizit angeben.

- v2: [CredentialProviderChain](#) In Node.js werden Anmeldeinformationen aus Quellen in der folgenden Reihenfolge aufgelöst:
 - [Umgebungsvariable](#)

- [Datei mit gemeinsam genutzten Anmeldeinformationen](#)
- [Anmeldeinformationen für den ECS-Container](#)
- [Externer Prozess wird gestartet](#)
- [OIDC-Token aus der angegebenen Datei](#)
- [EC2Amazon-Instanz-Metadaten](#)

Wenn einer der oben genannten Anmeldeinformationsanbieter die AWS Anmeldeinformationen nicht auflösen kann, fällt die Kette auf den nächsten Anbieter zurück, bis ein gültiger Berechtigungsnachweis gelöst ist, und die Kette gibt einen Fehler aus, wenn alle Anbieter ausfallen.

In Browser- und React Native-Laufzeiten ist die Anmeldeinformationskette leer und die Anmeldeinformationen müssen explizit festgelegt werden.

- v3: [defaultProvider](#). Die Quellen der Anmeldeinformationen und die Reihenfolge der Fallbacks ändern sich in Version 3 nicht. [Es unterstützt AWS IAM Identity Center auch Anmeldeinformationen.](#)

Temporäre Anmeldeinformationen

- v2: [ChainableTemporaryCredentials](#) steht für temporäre Anmeldeinformationen, die von abgerufen wurden `AWS.STS`. Ohne zusätzliche Parameter werden die Anmeldeinformationen aus dem `AWS.STS.getSessionToken()` Vorgang abgerufen. Wenn eine IAM-Rolle bereitgestellt wird, wird der `AWS.STS.assumeRole()` Vorgang stattdessen verwendet, um Anmeldeinformationen für die Rolle abzurufen. `AWS.ChainableTemporaryCredentials` unterscheidet sich von `AWS.TemporaryCredentials` der Art und Weise, wie `MasterCredentials` und Aktualisierungen behandelt werden. `AWS.ChainableTemporaryCredentials` aktualisiert abgelaufene Anmeldeinformationen mithilfe der vom Benutzer übergebenen `MasterCredentials`, um die Verkettung von STS-Anmeldeinformationen zu unterstützen. Reduziert jedoch die `MasterCredentials` während der Instanziierung `AWS.TemporaryCredentials` rekursiv, sodass Anmeldeinformationen, für die temporäre Anmeldeinformationen erforderlich sind, nicht aktualisiert werden können.

Das Original [TemporaryCredentials](#) wurde in Version 2 zugunsten von veraltet. `ChainableTemporaryCredentials`

- v3: [fromTemporaryCredentials](#) Sie können vom `@aws-sdk/credential-providers` Paket `fromTemporaryCredentials()` aus anrufen. Ein Beispiel:

```
import { FooClient } from "@aws-sdk/client-foo";
import { fromTemporaryCredentials } from "@aws-sdk/credential-providers"; // ES6
import
// const { FooClient } = require("@aws-sdk/client-foo");
// const { fromTemporaryCredentials } = require("@aws-sdk/credential-providers"); //
CommonJS import

const sourceCredentials = {
  // A credential can be a credential object or an async function that returns a
  credential object
};
const client = new FooClient({
  credentials: fromTemporaryCredentials({
    masterCredentials: sourceCredentials,
    params: { RoleArn },
  }),
});
```

Amazon Cognito Cognito-Identitätsanmeldedaten

Laden Sie Anmeldeinformationen aus dem Amazon Cognito Identity Service, der normalerweise in Browsern verwendet wird.

- v2: [CognitoIdentityCredentials](#) Stellt Anmeldeinformationen dar, die mithilfe des Amazon Cognito Identity Service von STS Web Identity Federation abgerufen wurden.
- v3: [Cognito Identity Credential Provider](#) Das [@aws/credential-providersPaket](#) stellt zwei Funktionen des Anmeldeinformationsanbieters bereit, von denen [fromCognitoIdentity](#) eine Identitäts-ID und Aufrufe [fromCognitoIdentityPool](#) entgegennimmt `cognitoIdentity: GetCredentialsForIdentity`, während die andere eine Identitätspool-ID verwendet, beim ersten Aufruf `cognitoIdentity: GetId` aufruft und dann aufruft. `fromCognitoIdentity` Nachfolgende Aufrufe des letzteren werden nicht erneut aufgerufen. `GetId`

Der Anbieter implementiert den im [Amazon Cognito Developer Guide](#) beschriebenen „Simplified Flow“. Der „Classic Flow“, bei dem zuerst angerufen `cognito: GetOpenIdToken` und dann angerufen `sts: AssumeRoleWithWebIdentity` wird, wird nicht unterstützt. Bitte senden Sie uns eine [Funktionsanfrage](#), falls Sie diese benötigen.

```
// fromCognitoIdentityPool example
import { fromCognitoIdentityPool } from "@aws-sdk/credential-providers"; // ES6
import
// const { fromCognitoIdentityPool } = require("@aws-sdk/credential-providers"); //
CommonJS import

const client = new FooClient({
  region: "us-east-1",
  credentials: fromCognitoIdentityPool({
    clientConfig: cognitoIdentityClientConfig, // Optional
    identityPoolId: "us-east-1:1699ebc0-7900-4099-b910-2df94f52a030",
    customRoleArn: "arn:aws:iam::1234567890:role/MYAPP-CognitoIdentity", // Optional
    logins: {
      // Optional
      "graph.facebook.com": "FBTOKEN",
      "www.amazon.com": "AMAZONTOKEN",
      "api.twitter.com": "TWITTERTOKEN",
    },
  }),
});
```

```
// fromCognitoIdentity example
import { fromCognitoIdentity } from "@aws-sdk/credential-providers"; // ES6 import
// const { fromCognitoIdentity } = require("@aws-sdk/credential-provider-cognito-
identity"); // CommonJS import

const client = new FooClient({
  region: "us-east-1",
  credentials: fromCognitoIdentity({
    clientConfig: cognitoIdentityClientConfig, // Optional
    identityId: "us-east-1:128d0a74-c82f-4553-916d-90053e4a8b0f",
    customRoleArn: "arn:aws:iam::1234567890:role/MYAPP-CognitoIdentity", // Optional
    logins: {
      // Optional
      "graph.facebook.com": "FBTOKEN",
      "www.amazon.com": "AMAZONTOKEN",
      "api.twitter.com": "TWITTERTOKEN",
    },
  }),
});
```

Anmeldedaten für Amazon EC2 Metadata (IMDS)

Stellt Anmeldeinformationen dar, die vom Metadaten-Service auf einer EC2 Amazon-Instance empfangen wurden.

- v2: [EC2MetadataCredentials](#)
- v3: [fromInstanceMetadata](#). Erstellt einen Anmeldeinformationsanbieter, der Anmeldeinformationen vom Amazon EC2 Instance Metadata Service bezieht.

```
import { fromInstanceMetadata } from "@aws-sdk/credential-providers"; // ES6 import
// const { fromInstanceMetadata } = require("@aws-sdk/credential-providers"); //
CommonJS import

const client = new FooClient({
  credentials: fromInstanceMetadata({
    maxRetries: 3, // Optional
    timeout: 0, // Optional
  }),
});
```

Amazon ECS-Anmeldeinformationen

Stellt Anmeldeinformationen dar, die von einer angegebenen URL empfangen wurden. Dieser Anbieter fordert temporäre Anmeldeinformationen von einer URI an, die in der `AWS_CONTAINER_CREDENTIALS_FULL_URI` Umgebungsvariablen `AWS_CONTAINER_CREDENTIALS_RELATIVE_URI` oder angegeben ist.

- v2: `ECSCredentials` oder [RemoteCredentials](#)
- v3: [fromContainerMetadata](#). Erstellt einen Anmeldeinformationsanbieter, der Anmeldeinformationen vom Amazon ECS Container Metadata Service bezieht.

```
import { fromContainerMetadata } from "@aws-sdk/credential-providers"; // ES6 import

const client = new FooClient({
  credentials: fromContainerMetadata({
    maxRetries: 3, // Optional
    timeout: 0, // Optional
  }),
});
```


Anmeldeinformationen für das Dateisystem

- v2: [FileSystemCredentials](#). Stellt Anmeldeinformationen aus einer JSON-Datei auf der Festplatte dar.
- v3: Veraltet. Sie können die JSON-Datei explizit lesen und dem Client zur Verfügung stellen. Bitte senden Sie uns eine [Funktionsanfrage](#), falls Sie diese benötigen.

Anbieter von SAML-Anmeldeinformationen

- v2: [SAMLCredentials](#) Stellt Anmeldeinformationen dar, die von der STS-SAML-Unterstützung abgerufen wurden.
- v3: Nicht verfügbar. Bitte senden Sie uns eine [Funktionsanfrage](#), falls Sie diese benötigen.

Anmeldeinformationen für die gemeinsame Anmeldeinformationsdatei

Lädt Anmeldeinformationen aus der Datei mit gemeinsam genutzten Anmeldeinformationen (standardmäßig ~/.aws/credentials oder definiert durch die AWS_SHARED_CREDENTIALS_FILE Umgebungsvariable). Diese Datei wird in verschiedenen AWS SDKs Tools unterstützt. Weitere Informationen finden Sie im [Dokument mit den gemeinsam genutzten Konfigurations- und Anmeldedaten](#).

- v2: [SharedIniFileCredentials](#)
- v3: [fromIni](#)

```
import { fromIni } from "@aws-sdk/credential-providers";
// const { fromIni } from("@aws-sdk/credential-providers");

const client = new FooClient({
  credentials: fromIni({
    configFilePath: "~/.aws/config", // Optional
    filepath: "~/.aws/credentials", // Optional
    mfaCodeProvider: async (mfaSerial) => {
      // implement a pop-up asking for MFA code
      return "some_code";
    }, // Optional
    profile: "default", // Optional
    clientConfig: { region }, // Optional
  }),
```

```
});
```

Anmeldeinformationen für die Web-Identität

Ruft Anmeldeinformationen mithilfe des OIDC-Tokens aus einer Datei auf der Festplatte ab. Wird häufig in Amazon EKS verwendet.

- Version 2: [TokenFileWebIdentityCredentials](#)
- v3: [fromTokenFile](#)

```
import { fromTokenFile } from "@aws-sdk/credential-providers"; // ES6 import
// const { fromTokenFile } from("@aws-sdk/credential-providers"); // CommonJS import

const client = new FooClient({
  credentials: fromTokenFile({
    // Optional. If skipped, read from `AWS_ROLE_ARN` environmental variable
    roleArn: "arn:xxxx",
    // Optional. If skipped, read from `AWS_ROLE_SESSION_NAME` environmental variable
    roleSessionName: "session:a",
    // Optional. STS client config to make the assume role request.
    clientConfig: { region },
  }),
});
```

Anmeldeinformationen für Web Identity Federation

Ruft Anmeldeinformationen von der STS-Unterstützung für Web Identity Federation ab.

- v2: [WebIdentityCredentials](#)
- v3: [fromWebToken](#)

```
import { fromWebToken } from "@aws-sdk/credential-providers"; // ES6 import
// const { fromWebToken } from("@aws-sdk/credential-providers"); // CommonJS import

const client = new FooClient({
  credentials: fromWebToken({
    // Optional. If skipped, read from `AWS_ROLE_ARN` environmental variable
    roleArn: "arn:xxxx",
```

```
// Optional. If skipped, read from `AWS_ROLE_SESSION_NAME` environmental variable
roleSessionName: "session:a",
// Optional. STS client config to make the assume role request.
clientConfig: { region },
}),
});
```

Überlegungen zu Amazon S3

Mehrteiliger Amazon S3 S3-Upload

In Version 2 enthält der Amazon S3-Client einen [upload\(\)](#)Vorgang, der das Hochladen großer Objekte mit der [von Amazon S3 angebotenen Funktion zum mehrteiligen Hochladen](#) unterstützt.

In Version 3 ist das [@aws-sdk/lib-storage](#)Paket verfügbar. Es unterstützt alle Funktionen, die im `upload()` v2-Betrieb angeboten werden, und unterstützt sowohl Node.js als auch die Browser-`Runtime`.

Vorsignierte Amazon S3 S3-URL

In Version 2 enthält der Amazon S3-Client die [getSignedUrlPromise\(\)](#)Operationen [getSignedUrl\(\)](#)und zum Generieren einer URL, mit der Benutzer Objekte von Amazon S3 hoch- oder herunterladen können.

In Version 3 ist das [@aws-sdk/s3-request-presigner](#)Paket verfügbar. Dieses Paket enthält die Funktionen für beide `getSignedUrl()` `getSignedUrlPromise()` Operationen. In diesem [Blogbeitrag](#) werden die Details dieses Pakets erörtert.

Amazon S3 S3-Regionsumleitungen

Wenn eine falsche Region an den Amazon S3-Client übergeben wird und ein darauffolgender Fehler `PermanentRedirect` (Status 301) ausgelöst wird, unterstützt der Amazon S3 S3-Client in Version 3 Regionsumleitungen (früher bekannt als Amazon S3 Global Client in Version 2). Sie können das [followRegionRedirects](#)Flag in der Client-Konfiguration verwenden, damit der Amazon S3 S3-Client regionalen Weiterleitungen folgt und seine Funktion als globaler Client unterstützt.

Note

Beachten Sie, dass diese Funktion zu zusätzlicher Latenz führen kann, da fehlgeschlagene Anfragen mit einer korrigierten Region erneut versucht werden, wenn ein

PermanentRedirect Fehler mit dem Status 301 empfangen wird. Diese Funktion sollte nur verwendet werden, wenn Sie die Region Ihrer Buckets nicht im Voraus kennen.

Amazon S3 S3-Streaming und gepufferte Antworten

Das v3-SDK zieht es vor, potenziell große Antworten nicht zu puffern. Dies tritt häufig bei der Amazon S3 GetObject S3-Operation auf, bei der Buffer in Version 2 a zurückgegeben wurde, Stream in Version 3 jedoch a zurückgegeben wurde.

Für Node.js müssen Sie den Stream verwenden oder den Client oder seinen Request-Handler löschen, um die Verbindungen für neuen Datenverkehr offen zu halten, indem Sie Sockets freigeben.

```
// v2
const get = await s3.getObject({ ... }).promise(); // this buffers consumes the stream
already.
```

```
// v3, consume the stream to free the socket
const get = await s3.getObject({ ... }); // object .Body has unconsumed stream
const str = await get.Body.transformToString(); // consumes the stream

// other ways to consume the stream include writing it to a file,
// passing it to another consumer like an upload, or buffering to
// a string or byte array.
```

Weitere Informationen finden Sie im Abschnitt zur [Socket-Erschöpfung](#).

DynamoDB-Dokumentenclient

Grundlegende Verwendung des DynamoDB-Dokumentenclients in Version 3

- In Version 2 können Sie die [AWS.DynamoDB.DocumentClient](#) Klasse verwenden, um DynamoDB APIs mit systemeigenen JavaScript Typen wie Array, Number und Object aufzurufen. Es vereinfacht somit die Arbeit mit Elementen in Amazon DynamoDB, indem der Begriff der Attributwerte weggelassen wird.
- In Version 3 ist der entsprechende [@aws-sdk/lib-dynamodb](#) Client verfügbar. Es ähnelt normalen Service-Clients aus dem v3-SDK, mit dem Unterschied, dass es einen einfachen DynamoDB-Client in seinem Konstruktor verwendet.

Beispiel:

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb"; // ES6 import
// const { DynamoDBClient } = require("@aws-sdk/client-dynamodb"); // CommonJS import
import { DynamoDBDocumentClient, PutCommand } from "@aws-sdk/lib-dynamodb"; // ES6
import
// const { DynamoDBDocumentClient, PutCommand } = require("@aws-sdk/lib-dynamodb"); //
CommonJS import

// Bare-bones DynamoDB Client
const client = new DynamoDBClient({});

// Bare-bones document client
const ddbDocClient = DynamoDBDocumentClient.from(client); // client is DynamoDB client

await ddbDocClient.send(
  new PutCommand({
    TableName,
    Item: {
      id: "1",
      content: "content from DynamoDBDocumentClient",
    },
  })
);
```

UndefinedWerte in beim Marshalling

- In Version 2 wurden undefined Werte in Objekten beim Marshalling-Prozess für DynamoDB automatisch weggelassen.
- In Version 3 `@aws-sdk/lib-dynamodb` hat sich das Standard-Marshalling-Verhalten geändert: Objekte mit undefined Werten werden nicht mehr weggelassen. Um der Funktionalität von v2 gerecht zu werden, müssen Entwickler den `removeUndefinedValues` Wert `true` im `DynamoDB Document Client` explizit auf setzen. `marshallOptions`

Beispiel:

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { DynamoDBDocumentClient, PutCommand } from "@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
```

```
// The DynamoDBDocumentClient is configured to handle undefined values properly
const ddbDocClient = DynamoDBDocumentClient.from(client, {
  marshallOptions: {
    removeUndefinedValues: true
  }
});

await ddbDocClient.send(
  new PutCommand({
    TableName,
    Item: {
      id: "123",
      content: undefined // This value will be automatically omitted.
      array: [1, undefined], // The undefined value will be automatically omitted.
      map: { key: undefined }, // The "key" will be automatically omitted.
      set: new Set([1, undefined]), // The undefined value will be automatically
      omitted.
    }
  });
});
```

Weitere Beispiele und Konfigurationen sind im [Paket README](#) verfügbar.

Kellner und Unterzeichner

Auf dieser Seite wird die Verwendung von Kellnern und Unterzeichnern in Version 3 beschrieben.

AWS SDK für JavaScript

Waiter

In Version 2 sind alle Kellner an die Klasse Service Client gebunden, und Sie müssen in der Eingabe des Kellners angeben, auf welchen festgelegten Status der Kunde warten wird. Beispielsweise müssen Sie anrufen, um darauf [waitFor\("bucketExists"\)](#) zu warten, dass ein neu erstellter Bucket bereit ist.

In Version 3 müssen Sie keine Kellner importieren, wenn Ihre Anwendung keinen benötigt. Darüber hinaus können Sie nur den Kellner importieren, den Sie benötigen, um auf den gewünschten Bundesstaat zu warten. So können Sie Ihre Paketgröße reduzieren und die Leistung verbessern. Hier ist ein Beispiel dafür, wie Sie darauf warten, dass der Bucket nach der Erstellung bereit ist:

```
import { S3Client, CreateBucketCommand, waitUntilBucketExists } from "@aws-sdk/client-s3"; // ES6 import
```

```
// const { S3Client, CreateBucketCommand, waitUntilBucketExists } = require("@aws-sdk/
client-s3"); // CommonJS import

const Bucket = "BUCKET_NAME";
const client = new S3Client({ region: "REGION" });
const command = new CreateBucketCommand({ Bucket });

await client.send(command);
await waitUntilBucketExists({ client, maxWaitTime: 60 }, { Bucket });
```

Alles zur Konfiguration der Kellner findest du im [Blogbeitrag von waiters in the AWS SDK für JavaScript v3](#).

CloudFront Amazon-Unterzeichner

In Version 2 können Sie die Anfrage für den Zugriff auf eingeschränkte CloudFront Amazon-Distributionen mit [AWS.CloudFront.Signer](#) unterschreiben.

In Version 3 stehen Ihnen dieselben Dienstprogramme zur Verfügung, die im [@aws-sdk/cloudfront-signer](#) Paket enthalten sind.

Amazon RDS-Unterzeichner

In Version 2 können Sie das Authentifizierungstoken für eine Amazon RDS-Datenbank mithilfe von [AWS.RDS.Signer](#) generieren.

In Version 3 ist die ähnliche Dienstprogrammklasse im [@aws-sdk/rds-signer](#) Paket verfügbar.

Amazon Polly Signer

In Version 2 können Sie eine signierte URL für die vom Amazon Polly Polly-Service synthetisierte Sprache mit generieren. [AWS.Polly.Presigner](#)

In Version 3 ist eine ähnliche Hilfsfunktion im [@aws-sdk/polly-request-presigner](#) Paket verfügbar.

Hinweise zu bestimmten Servicekunden

AWS Lambda

Der Antworttyp für Lambda-Aufrufe unterscheidet sich in v2 und v3.

```
// v2
```

```
import { Lambda } from "@aws-sdk/client-lambda";
import AWS from "aws-sdk";

const lambda = new AWS.Lambda({ REGION });
const invoke = await lambda.invoke({
  FunctionName: "echo",
  Payload: JSON.stringify({ message: "hello" }),
}).promise();

// in v2, Lambda::invoke::Payload is automatically converted to string via a
// specific code customization.
const payloadIsString = typeof invoke.Payload === "string";
console.log("Invoke response payload type is string:", payloadIsString);

const payloadObject = JSON.parse(invoke.Payload);
console.log("Invoke response object", payloadObject);
```

```
// v3
const lambda = new Lambda({ REGION });
const invoke = await lambda.invoke({
  FunctionName: "echo",
  Payload: JSON.stringify({ message: "hello" }),
});

// in v3, Lambda::invoke::Payload is not automatically converted to a string.
// This is to reduce the number of customizations that create inconsistent behaviors.
const payloadIsByteArray = invoke.Payload instanceof Uint8Array;
console.log("Invoke response payload type is Uint8Array:", payloadIsByteArray);

// To maintain the old functionality, only one additional method call is needed:
// v3 adds a method to the Uint8Array called transformToString.
const payloadObject = JSON.parse(invoke.Payload.transformToString());
console.log("Invoke response object", payloadObject);
```

Amazon SQS

MD5 Prüfsumme

Um die Berechnung der MD5 Prüfsummen von Nachrichtentexten zu überspringen, setzen Sie den Wert für das Konfigurationsobjekt *md5* auf *false*. Andernfalls berechnet das SDK standardmäßig die Prüfsumme für das Senden von Nachrichten und validiert die Prüfsumme für abgerufene Nachrichten.


```
// Example: Skip MD5 checksum in Amazon SQS
import { SQS } from "@aws-sdk/client-sqs";

new SQS({
  md5: false // note: only available in v3.547.0 and higher
});
```

Bei der Verwendung eines benutzerdefinierten Elements `QueueUrl` in Amazon SQS SQS-Vorgängen, bei dem dieser Parameter als Eingabeparameter verwendet wurde, war es in Version 2 möglich, einen benutzerdefinierten Wert anzugeben, der den `QueueUrl` Standardendpunkt des Amazon SQS SQS-Clients überschreiben würde.

Nachrichten aus mehreren Regionen

In Version 3 sollten Sie einen Client pro Region verwenden. Die AWS Region soll auf Client-Ebene initialisiert und nicht zwischen Anfragen geändert werden.

```
import { SQS } from "@aws-sdk/client-sqs";

const sqsClients = {
  "us-east-1": new SQS({ region: "us-east-1" }),
  "us-west-2": new SQS({ region: "us-west-2" }),
};

const queues = [
  { region: "us-east-1", url: "https://sqs.us-east-1.amazonaws.com/{AWS_ACCOUNT}/MyQueue" },
  { region: "us-west-2", url: "https://sqs.us-west-2.amazonaws.com/{AWS_ACCOUNT}/MyOtherQueue" },
];

for (const { region, url } of queues) {
  const params = {
    MessageBody: "Hello",
    QueueUrl: url,
  };
  await sqsClients[region].sendMessage(params);
}
```

Benutzerdefinierter Endpunkt

Wenn Sie in Version 3 einen benutzerdefinierten Endpunkt verwenden, d. h. einen, der sich von den standardmäßigen öffentlichen Amazon SQS SQS-Endpunkten unterscheidet, sollten Sie sowohl den Endpunkt auf dem Amazon SQS SQS-Client als auch das Feld festlegen. `QueueUrl`

```
import { SQS } from "@aws-sdk/client-sqs";

const sqs = new SQS({
  // client endpoint should be specified in v3 when not the default public SQS endpoint
  // for your region.
  // This is required for versions <= v3.506.0
  // This is optional but recommended for versions >= v3.507.0 (a warning will be
  // emitted)
  endpoint: "https://my-custom-endpoint:8000/",
});

await sqs.sendMessage({
  QueueUrl: "https://my-custom-endpoint:8000/1234567/MyQueue",
  Message: "hello",
});
```

Wenn Sie keinen benutzerdefinierten Endpunkt verwenden, müssen Sie ihn nicht `endpoint` auf dem Client einrichten.

```
import { SQS } from "@aws-sdk/client-sqs";

const sqs = new SQS({
  region: "us-west-2",
});

await sqs.sendMessage({
  QueueUrl: "https://sqs.us-west-2.amazonaws.com/1234567/MyQueue",
  Message: "hello",
});
```

Zusätzliche Unterlagen

Die folgende Tabelle enthält Links zu ergänzender Dokumentation, die Ihnen die Verwendung und das Verständnis von AWS SDK für JavaScript (v3) erleichtern soll.

Name	Hinweise
SDK-Clients	Informationen zur Initialisierung eines SDK-Clients und zu häufig verwendeten konfigurierten Konstruktorparametern.
Aktualisierung von Notes (2.x auf 3.x)	Informationen zum Upgrade von AWS SDK für JavaScript (v2).
Verwenden von AWS SDK für JavaScript (v3) auf Laufzeiten von AWS Lambda Node.js	Bewährte Methoden für die Arbeit AWS Lambda mit der AWS SDK für JavaScript (v3).
Leistung	Informationen darüber, wie das AWS SDK-Team die Leistung des SDK optimiert hat, sowie Tipps zur Konfiguration des SDK für einen effizienten Betrieb.
TypeScript	TypeScript Tipps und FAQs Hinweise zum AWS SDK für JavaScript (v3).
Behandlung von Fehlern	Tipps zum Umgang mit Fehlern im Zusammenhang mit AWS SDK für JavaScript (v3).
Effektive Praktiken	Allgemeine Empfehlungen für die Verwendung von AWS SDK für JavaScript (v3).

Dokumentenverlauf für AWS SDK für JavaScript Version 3

Dokumentverlauf

In der folgenden Tabelle werden die wichtigen Änderungen in der Version V3 AWS SDK für JavaScript ab dem 20. Oktober 2020 beschrieben. Für Benachrichtigungen über Aktualisierungen dieser Dokumentation können Sie einen [RSS-Feed](#) abonnieren.

Änderung	Beschreibung	Datum
TLS 1.3 wird jetzt auf allen AWS Service-API-Endpunkten in allen Regionen unterstützt	Die unterstützte TLS-Version und die Methode zur Protokollierung der TLS-Version wurden aktualisiert.	10. April 2025
Generierung von Clients mit dem @smithy /types	Der Inhalt wurde beim Generieren von Clients mit dem Paket @smithy /types aktualisiert.	15. Februar 2025
Schutz der Datenintegrität mit Prüfsummen	Der Inhalt wurde mit Einzelheiten zur automatischen Prüfsummenberechnung aktualisiert.	15. Januar 2025
Amazon S3 S3-Prüfsummen	Es wurde ein Abschnitt zur Verwendung flexibler Prüfsummen mit Amazon S3 hinzugefügt.	1. Januar 2025
Support für kontobasierte Endpunkte in DynamoDB	Die AWS SDK für JavaScript zusätzliche Unterstützung für kontobasierte Endpunkte in DynamoDB.	26. September 2024
Neues Thema zur SDK-Protokollierung	Es wurde ein Thema hinzugefügt, das beschreibt,	26. September 2024

wie API-Aufrufe protokolliert werden, die mit dem SDK für getätigt wurden. Es JavaScript enthält auch Informationen zur Verwendung von Middleware zum Protokollieren von Anfragen.

[Ankündigung](#)

Das obere Banner mit einer end-of-support Erinnerung für Internet Explorer 11 wurde aktualisiert.

23. September 2022

[Kleinere Updates](#)

Kleinere Aktualisierungen zur besseren Übersicht und zur Behebung defekter Links. Es wurden Links zur Sensibilisierung AWS SDKs und zum Referenzhandbuch für Tools hinzugefügt.

22. August 2022

[Erzwingen Sie eine TLS-Mindestversion](#)

Es wurden Informationen zu TLS 1.3 hinzugefügt.

31. März 2022

[Das Thema „Anmeldeinformationen in Node.js festlegen“ wurde aktualisiert](#)

Aktualisiertes Thema zum Einstellen von Anmeldeinformationen in Node.js für AWS SDK für JavaScript V3.

20. Oktober 2020

[Migrieren Sie zu v3](#)

Ein Thema zur Beschreibung der Migration zu AWS SDK für JavaScript Version 3 wurde hinzugefügt.

20. Oktober 2020

<u>Erste Schritte</u>	Die Themen für die ersten Schritte im Browser und die ersten Schritte mit Node.js für AWS SDK für JavaScript V3 wurden aktualisiert.	20. Oktober 2020
<u>Browser-Builder</u>	Informationen zu AWS Browser Builder wurden entfernt, da sie für AWS SDK für JavaScript Version 3 nicht erforderlich sind.	20. Oktober 2020
<u>Servicebeispiele für Amazon Transcribe aktualisiert</u>	Aktualisierte Amazon Transcribe-Servicebeispiele für AWS SDK für JavaScript V3.	20. Oktober 2020
<u>Servicebeispiele für Amazon Simple Notification Service aktualisiert</u>	Die Service-Beispiele von Amazon Simple Notification Service für AWS SDK für JavaScript Version 3 wurden aktualisiert.	20. Oktober 2020
<u>Servicebeispiele für Amazon Simple Email Service aktualisiert</u>	Die Service-Beispiele von Amazon Simple Email Service für AWS SDK für JavaScript Version 3 wurden aktualisiert.	20. Oktober 2020
<u>Amazon Redshift Redshift-Servicebeispiele aktualisiert</u>	Aktualisierte Amazon Redshift Redshift-Servicebeispiele für AWS SDK für JavaScript V3.	20. Oktober 2020
<u>Amazon Lex Lex-Serveisbeispiele aktualisiert</u>	Die Amazon Lex Lex-Serveisbeispiele für AWS SDK für JavaScript V3 wurden aktualisiert.	20. Oktober 2020

[AWS Elemental MediaConvert
Servicebeispiele aktualisiert](#)

AWS Elemental MediaConvert
Servicebeispiele für AWS SDK
für JavaScript V3 aktualisiert.

20. Oktober 2020

[AWS Lambda Servicebeispiele
aktualisiert](#)

AWS Lambda Servicebeispiele
für AWS SDK für JavaScript
V3 aktualisiert.

20. Oktober 2020

[AWS SDK für JavaScript
t Vorschau des V3-Entwic
kerhandbuches](#)

Veröffentlichte Vorabversion
des AWS SDK für JavaScript
V3 Developer Guide.

19. Oktober 2020