aws

Entwicklerhandbuch

Amazon Kinesis Data Streams

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

Amazon Kinesis Data Streams: Entwicklerhandbuch

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

Was ist Amazon Kinesis Data Streams? ... 1
Was kann ich mit Kinesis Data Streams machen? ... 1
Vorteile der Verwendung von Kinesis Data Streams ... 2
Zugehörige Services ... 3
Terminologie und Konzepte ... 4
Informieren Sie sich über die High-Level-Architektur von Kinesis Data Streams ... 4
Machen Sie sich mit der Terminologie von Kinesis Data Streams vertraut ... 5
Kinesis Data Stream ... 5
Datensatz ... 5
Kapazitätsmodus ... 5
Aufbewahrungszeitraum ... 5
Produzent ... 6
Konsument ... 6
Anwendung von Amazon Kinesis Data Streams ... 6
Shard ... 6
Partitionsschlüssel ... 7
Sequenznummer ... 7
Kinesis Client Library ... 7
Anwendungsname ... 8
Serverseitige Verschlüsselung ... 8
Kontingente und -Einschränkungen ... 9
Limits für API ... 12
KDS-Limits für die API der Steuerebene ... 12
Limits für die KDS-API der Datenebene ... 18
Erhöhung der Kontingente ... 21
Vollständige Voraussetzungen für die Einrichtung von Amazon Kinesis Data Streams ... 22
Melden Sie sich an für AWS ... 22
Bibliotheken und Tools herunterladen ... 22
Konfigurieren Sie Ihre Entwicklungsumgebung ... 23
Verwenden Sie die AWS CLI , um Amazon Kinesis Data Streams Streams-Operationen durchzuführen ... 24
Tutorial: Installation und Konfiguration von AWS CLI for Kinesis Data Streams ... 25
Installieren Sie das AWS CLI ... 25
Konfigurieren Sie das AWS CLI ... 26

Tutorial: Führen Sie grundlegende Kinesis Data Streams Streams-Operationen mit dem AWS CLI ... 26
Schritt 1: Erstellen Sie einen Stream ... 27
Schritt 2: Einen Datensatz erstellen ... 29
Schritt 3: Holen Sie sich den Datensatz ... 29
Schritt 4: Bereinigen ... 32
Tutorials für die ersten Schritte ... 34
Tutorial: Verarbeiten Sie Aktiendaten in Echtzeit mit KPL und KCL 2.x ... 34
Erfüllen der Voraussetzungen ... 35
Erstellen Sie einen Datenstream ... 36
Erstellen Sie eine IAM-Richtlinie und einen Benutzer ... 37
Laden Sie den Code herunter und erstellen Sie ihn ... 43
Implementieren Sie den Produzenten ... 44
Implementieren Sie den Verbraucher ... 48
(Optional) Erweitern Sie den Consumer ... 53
Bereinigen von Ressourcen ... 55
Tutorial: Verarbeiten Sie Aktiendaten in Echtzeit mit KPL und KCL 1.x ... 56
Erfüllen der Voraussetzungen ... 57
Erstellen Sie einen Datenstream ... 59
Erstellen Sie eine IAM-Richtlinie und einen Benutzer ... 60
Laden Sie den Implementierungscode herunter und erstellen Sie ihn ... 66
Implementieren Sie den Produzenten ... 67
Implementieren Sie den Verbraucher ... 71
(Optional) Erweitern Sie die Zahl der Verbraucher ... 75
Bereinigen von Ressourcen ... 77
Tutorial: Analysieren Sie Aktiendaten in Echtzeit mit Amazon Managed Service für Apache Flink ... 79
Voraussetzungen ... 80
Schritt 1: Einrichten eines -Kontos ... 80
Schritt 2: Richten Sie das ein AWS CLI ... 84
Schritt 3: Erstellen Sie eine Anwendung ... 86
Tutorial: Verwendung AWS Lambda mit Amazon Kinesis Data Streams ... 104
Verwenden Sie die AWS Streaming-Datenlösung für Amazon Kinesis ... 104
Kinesis-Datenstreams erstellen und verwalten ... 106
Wählen Sie den richtigen Modus zum Streamen ... 107
Was sind die verschiedenen Modi in Kinesis Data Streams? ... 107

Funktionen und Anwendungsfälle im On-Demand-Standardmodus 108
Funktionen und Anwendungsfälle des Advantage-Modus auf Abruf 110
Funktionen und Anwendungsfälle im Bereitstellungsmodus 111
Wechseln Sie zwischen den Modi 112
Erstellen Sie einen Stream mit dem AWS-Managementkonsole 113
Erstellen Sie einen Stream mit dem APIs 114
Den Kinesis Data Streams Streams-Client erstellen 114
Erstellen Sie den Stream 114
Aktualisieren Sie einen Stream 116
Verwenden der Konsole 116
Verwenden der API 117
Verwenden Sie den AWS CLI 118
Auflisten von Streams 118
Shards auflisten 119
Löschen Sie einen Stream 122
Einen Stream erneut teilen 123
Entscheiden Sie sich für eine Strategie für das Resharding 124
Teilen Sie einen Shard auf 125
Zwei Shards zusammenführen 127
Schließen Sie die Resharding-Aktion ab 128
Ändern Sie den Aufbewahrungszeitraum für Daten 131
Markieren Ihrer -Ressourcen mit Tags (Markierungen) 133
Lesen Sie die Grundlagen von Stichwörtern 133
Verfolgen Sie die Kosten mithilfe von Tagging 134
Verstehen Sie die Einschränkungen von Tags 135
Taggen Sie Streams mit der Kinesis Data Streams Streams-Konsole 136
Taggen Sie Streams mit dem AWS CLI 138
Taggen von Streams mithilfe der Kinesis Data Streams APIs 138
Taggen Sie Verbraucher mit dem AWS CLI 139
Taggen Sie Verbraucher, die Kinesis Data Streams verwenden APIs 139
Umgang mit großen Datensätzen 140
Aktualisiere deinen Stream, um große Datensätze zu verwenden 140
Optimiere deine Stream-Leistung mit großen Datensätzen 141
Reduzieren Sie die Drosselung bei großen Datensätzen 141
Verarbeiten Sie große Datensätze mit den Kinesis Data Streams APIs 141
AWS Komponenten, die mit großen Datensätzen kompatibel sind 142

Regionen, in denen große Datensätze unterstützt werden ... 146
Führen Sie Resilienztests durch mit AWS Fault Injection Service ... 148
Ausnahmefehler beim bereitgestellten Durchsatz ... 149
Exception-Fehler im abgelaufenen Iterator ... 152
Daten in Kinesis Data Streams schreiben ... 155
Entwickeln Sie Produzenten mithilfe der Amazon Kinesis Producer Library (KPL) ... 156
Sehen Sie sich die Rolle der KPL an ... 157
Machen Sie sich die Vorteile der Verwendung der KPL bewusst ... 158
Verstehen Sie, wann Sie die KPL nicht verwenden sollten ... 159
Installieren Sie die KPL ... 159
Migrieren Sie zu KPL 1.x ... 160
Umstellung auf Amazon Trust Services (ATS) -Zertifikate für die KPL ... 164
Von KPL unterstützte Plattformen ... 165
Die wichtigsten Konzepte von KPL ... 165
Integrieren Sie die KPL in den Herstellercode ... 168
Schreiben Sie mit der KPL in Ihren Kinesis-Datenstream ... 170
Konfigurieren Sie die KPL ... 172
Implementieren Sie die Deaggregation für Verbraucher ... 173
Verwenden Sie die KPL mit Amazon Data Firehose ... 177
Verwenden Sie die KPL mit der Schema Registry AWS Glue ... 177
Konfigurieren Sie die KPL-Proxykonfiguration ... 178
Richtlinie zum Lebenszyklus der KPL-Version ... 179
Entwickeln Sie Produzenten mithilfe der Kinesis Data Streams API mit dem AWS SDK für Java ... 180
Daten zu einem Stream hinzufügen ... 181
Interagieren Sie mit Daten mithilfe der AWS Glue Schema Registry ... 187
Schreiben Sie mit Kinesis Agent in Amazon Kinesis Data Streams ... 188
Erfüllen Sie die Voraussetzungen für Kinesis Agent ... 189
Herunterladen und Installieren des Agenten ... 189
Konfigurieren und starten Sie den Agenten ... 190
Geben Sie die Einstellungen für die Agentenkonfiguration an ... 191
Überwachen Sie mehrere Dateiverzeichnisse und schreiben Sie in mehrere Streams ... 195
Verwenden Sie den Agenten zur Vorverarbeitung von Daten ... 196
Verwenden Sie CLI-Befehle für Agenten ... 201
Häufig gestellte Fragen ... 201
Schreiben Sie mithilfe anderer AWS Dienste in Kinesis Data Streams ... 203

Schreiben Sie in Kinesis Data Streams mit AWS Amplify ... 203
Schreiben Sie mit Amazon Aurora in Kinesis Data Streams ... 204
Schreiben Sie mit Amazon in Kinesis Data Streams CloudFront ... 204
Schreiben Sie mithilfe von Amazon CloudWatch Logs in Kinesis Data Streams ... 204
Schreiben Sie mit Amazon Connect in Kinesis Data Streams ... 204
Schreiben Sie in Kinesis Data Streams mit AWS Database Migration Service ... 205
Schreiben Sie mit Amazon DynamoDB in Kinesis Data Streams ... 205
Schreiben Sie mit Amazon in Kinesis Data Streams EventBridge ... 205
Schreiben Sie in Kinesis Data Streams mit AWS IoT Core ... 206
Schreiben Sie mit Amazon Relational Database Service in Kinesis Data Streams ... 206
Schreiben Sie mit Amazon Pinpoint in Kinesis Data Streams ... 206
Schreiben Sie mithilfe der Amazon Quantum Ledger Database (Amazon QLDB) in Kinesis Data Streams ... 206
Schreiben Sie mithilfe von Integrationen von Drittanbietern in Kinesis Data Streams ... 207
Apache Flink ... 207
Fluentd ... 207
Debezium ... 208
Orakel GoldenGate ... 208
Kafka Connect ... 208
Adobe Experience ... 208
Striim ... 208
Problembehandlung bei Kinesis Data Streams Streams-Produzenten ... 208
Meine Producer-Anwendung schreibt langsamer als erwartet ... 209
Ich erhalte einen unautorisierten KMS-Masterkey-Berechtigungsfehler ... 211
Beheben Sie andere häufig auftretende Probleme von Herstellern ... 211
Optimieren Sie Kinesis Data Streams Streams-Produzenten ... 212
Passen Sie das Verhalten von KPL-Wiederholungen und das Verhalten bei der Ratenbegrenzung an ... 212
Wenden Sie bewährte Methoden auf die KPL-Aggregation an ... 213
Daten aus Kinesis Data Streams lesen ... 215
Entwickeln Sie verbesserte Fan-Out-Verbraucher mit dediziertem Durchsatz ... 216
Unterschiede zwischen einem Verbraucher mit gemeinsamem Durchsatz und einem Verbraucher mit erweitertem Fan-Out-Anschluss ... 218
Unterstützte Regionen für bis zu 50 erweiterte Fan-Out-Nutzer (nur On-Demand-Vorteil) ... 146
Verwalten Sie Kunden mit verbesserter Fangemeinde ... 221
Verwenden Sie den Data Viewer in der Kinesis-Konsole ... 223

Fragen Sie Ihre Datenströme in der Kinesis-Konsole ab 224
Verwenden Sie die Kinesis-Clientbibliothek 224
Was ist die Kinesis Client Library? 224
Die wichtigsten Funktionen und Vorteile von KCL 225
KCL-Konzepte 226
DynamoDB-Metadatentabellen und Lastenausgleich in KCL 227
Entwickeln Sie Verbraucher mit KCL 232
Multistream-Verarbeitung mit KCL 244
Verwenden Sie die AWS Glue Schemaregistrierung mit KCL 247
Für KCL-Anwendungen für Privatanwender sind IAM-Berechtigungen erforderlich 247
KCL-Konfigurationen 254
Lebenszyklus-Richtlinie für KCL-Versionen 274
Migrieren Sie von früheren KCL-Versionen 275
Dokumentation zu früheren KCL-Versionen 291
Entwickeln Sie Verbraucher mit dem AWS SDK für Java 378
Entwickeln Sie Verbraucher mit gemeinsamem Durchsatz mit dem AWS SDK für Java 379
Entwickeln Sie mehr Fan-Out-Nutzer mit dem AWS SDK für Java 385
Interagieren Sie mit Daten mithilfe der Schema Registry AWS Glue 388
Entwickeln Sie Verbraucher mit AWS Lambda 388
Entwickeln Sie Kunden mithilfe von Managed Service für Apache Flink 389
Entwickeln Sie Verbraucher mithilfe von Amazon Data Firehose 389
Verwenden Sie andere AWS Dienste, um Daten aus Kinesis Data Streams zu lesen 389
Daten aus Kinesis Data Streams mit Amazon EMR lesen 390
Daten aus Kinesis Data Streams mit Amazon EventBridge Pipes lesen 390
Daten aus Kinesis Data Streams lesen mit AWS Glue 390
Daten aus Kinesis Data Streams mit Amazon Redshift lesen 391
Lesen Sie mithilfe von Integrationen von Drittanbietern aus Kinesis Data Streams 391
Apache Flink 391
Adobe Experience Platform 392
Apache Druid 392
Apache Spark 392
Databricks 392
Kafka Confluent Plattform 393
Kinesumer 393
Talend 393
Problembehandlung bei Kinesis Data Streams Streams-Verbrauchern 393

Kompilierungsfehler mit dem Konstruktor LeaseManagementConfig 394
Einige Kinesis Data Streams Streams-Datensätze werden bei der Verwendung der Kinesis Client Library übersprungen 395
Datensätze, die zu demselben Shard gehören, werden gleichzeitig von verschiedenen Datensatzprozessoren verarbeitet 395
Die Verbraucheranwendung liest langsamer als erwartet 396
GetRecords gibt ein leeres Datensatz-Array zurück, auch wenn der Stream Daten enthält .. 397
Der Shard-Iterator läuft unerwartet ab 398
Die Verarbeitung von Verbraucherdaten hinkt hinterher 398
Fehler beim Zugriff auf einen nicht autorisierten KMS-Schlüssel 399
DynamoDbException: Der im Aktualisierungsausdruck angegebene Dokumentpfad ist für die Aktualisierung ungültig 400
Beheben Sie andere häufig auftretende Probleme von Verbrauchern 400
Optimieren Sie Kinesis Data Streams Streams-Nutzer 400
Verbessern Sie die Verarbeitung mit niedriger Latenz 401
Verarbeiten Sie serialisierte Daten AWS Lambda mithilfe der Amazon Kinesis Producer Library 402
Verwenden Sie Resharding, Skalierung und Parallelverarbeitung, um die Anzahl der Shards zu ändern 402
Behandeln Sie doppelte Datensätze 404
Kümmere dich um das Starten, Herunterfahren und Drosseln 407
Kinesis Data Streams überwachen 409
Überwachen Sie den Kinesis Data Streams Streams-Dienst mit CloudWatch 409
Dimensionen und Metriken von Amazon Kinesis Data Streams 410
Greifen Sie auf CloudWatch Amazon-Metriken für Kinesis Data Streams zu 427
Überwachen Sie den Zustand des Kinesis Data Streams Streams-Agenten mit CloudWatch 428
Überwachen Sie mit CloudWatch 428
API-Aufrufe von Amazon Kinesis Data Streams protokollieren mit AWS CloudTrail 429
Kinesis Data Streams Streams-Informationen in CloudTrail 429
Beispiel: Einträge in der Kinesis Data Streams Streams-Protokolldatei 431
Überwachen Sie die KCL mit CloudWatch 435
Metriken und Namespace 435
Metrische Ebenen und Dimensionen 436
Konfiguration der Metrik 437
Liste der Metriken 437
Überwachen Sie die KPL mit CloudWatch 455

Metriken, Dimensionen und Namespaces ... 456
Metrikebene und Granularität ... 456
Lokaler Zugriff und CloudWatch Amazon-Upload ... 457
Liste der Metriken ... 458
Sicherheit ... 463
Datenschutz in Kinesis Data Streams ... 464
Was ist serverseitige Verschlüsselung für Kinesis Data Streams? ... 464
Überlegungen zu Kosten, Regionen und Leistung ... 466
Wie fange ich mit serverseitiger Verschlüsselung an? ... 468
Erstellen und verwenden Sie benutzergenerierte KMS-Schlüssel ... 469
Berechtigungen zur Verwendung von benutzergenerierten KMS-Schlüsseln ... 469
Überprüfen Sie die KMS-Schlüsselberechtigungen und beheben Sie Fehler ... 472
Verwenden Sie Kinesis Data Streams mit Schnittstellen-VPC-Endpunkten ... 472
Steuern des Zugriffs auf Kinesis Data Streams Streams-Ressourcen mithilfe von IAM ... 476
Richtliniensyntax ... 477
Aktionen für Kinesis Data Streams ... 478
Amazon-Ressourcennamen (ARNs) für Kinesis Data Streams ... 479
Beispielrichtlinien für Kinesis Data Streams ... 479
Teilen Sie Ihren Datenstream mit einem anderen Konto ... 482
Konfigurieren Sie eine AWS Lambda Funktion zum Lesen aus Kinesis Data Streams in einem anderen Konto ... 488
Teilen Sie den Zugriff mithilfe ressourcenbasierter Richtlinien ... 488
Compliance-Validierung ... 491
Resilienz in Kinesis Data Streams ... 491
Notfallwiederherstellung in Kinesis Data Streams ... 492
Infrastruktursicherheit ... 493
Bewährte Sicherheitsmethoden für Kinesis Data Streams ... 493
Implementieren des Zugriffs mit geringsten Berechtigungen ... 494
Verwenden von IAM-Rollen ... 494
Implementieren Sie serverseitige Verschlüsselung in abhängigen Ressourcen ... 494
Wird CloudTrail zur Überwachung von API-Aufrufen verwendet ... 495
Arbeitet mit AWS SDKs ... 496
Codebeispiele ... 498
Grundlagen ... 499
Kennenlernen der Grundlagen ... 499
Aktionen ... 503

Serverless-Beispiele 561
Aufrufen einer Lambda-Funktion über einen Kinesis-Auslöser 562
Melden von Batch-Elementfehlern für Lambda-Funktionen mit einem Kinesis-Auslöser 572
Dokumentverlauf 587
..... dxci

Was ist Amazon Kinesis Data Streams?

Sie können Amazon Kinesis Data Streams zum Erfassen und Verarbeiten großer Streams von Datensätzen in Echtzeit nutzen. Sie können Datenverarbeitungsanwendungen erstellen, die als Anwendungen von Kinesis Data Streams bezeichnet werden. Eine typische Anwendung von Kinesis Data Streams liest Daten aus einem Datenstrom als Datensätze. Diese Anwendungen können die Kinesis Client Library verwenden und auf EC2 Amazon-Instances ausgeführt werden. Sie können die verarbeiteten Datensätze an Dashboards senden, sie zum Erstellen von Warnmeldungen, zum dynamischen Ändern von Preis- und Werbestrategien nutzen oder Daten an verschiedene andere AWS -Services senden. Informationen zu den Features und Preisen von Kinesis Data Streams finden Sie unter Amazon Kinesis Data Streams.

Kinesis Data Streams ist zusammen mit Firehose, Kinesis Video Streams und Managed Service for Apache Flink Teil der Kinesis-Streaming-Datenplattform.

Weitere Informationen zu AWS Big-Data-Lösungen finden Sie unter Big Data auf. AWS Weitere Informationen zu AWS -Streaming-Datenlösungen finden Sie unter Was sind Streaming-Daten?.

Themen

- Was kann ich mit Kinesis Data Streams machen?
- Vorteile der Verwendung von Kinesis Data Streams
- Zugehörige Services

Was kann ich mit Kinesis Data Streams machen?

Sie können Kinesis Data Streams für die schnelle und kontinuierliche Aufnahme und Aggregation von Daten verwenden. Der verwendete Datentyp kann Protokolldaten zur IT-Infrastruktur, Anwendungsprotokolle, Data-Feeds von sozialen Medien, Marktdaten-Feeds sowie Web-Clickstream-Daten einschließen. Da die Reaktionszeit für die Aufnahme und Verarbeitung der Daten in Echtzeit erfolgt, ist die Verarbeitung in der Regel ein leichtgewichtiger Prozess.

Es folgen typische Szenarien für den Einsatz von Kinesis Data Streams:

Beschleunigte Protokoll- und Datenfeedaufnahme und -verarbeitung

> Produzenten können Daten direkt in einen Stream einleiten. Sie können beispielsweise System- und Anwendungsprotokolle übertragen. Diese stehen dann innerhalb von Sekunden für die

Verarbeitung zur Verfügung. Dadurch wird verhindert, dass die Protokolldaten verloren gehen, wenn Front-End-Server oder Anwendungsserver abstürzen. Kinesis Data Streams unterstützt eine beschleunigte Datenfeedaufnahme, da die Daten auf den Servern nicht zu Stapeln zusammengefasst werden, bevor sie zur Aufnahme weitergeleitet werden.

Echtzeitmetriken und -berichte

Sie können Daten, die in Kinesis Data Streams erfasst wurden, für eine einfache Datenanalyse und Berichterstellung in Echtzeit verwenden. So kann Ihre Datenverarbeitungsanwendung beispielsweise während des Daten-Streamings an Metriken und Berichten für System- und Anwendungsprotokolle arbeiten und muss nicht auf einzelne Datenpakete warten.

Echtzeitdatenanalysen

Hier werden die Vorteile von Echtzeitdaten mit der parallelen Verarbeitung kombiniert. So können Sie Website-Clickstreams in Echtzeit verarbeiten und anschließend die Benutzerfreundlichkeit der Website mit verschiedenen Anwendungen von Kinesis Data Streams analysieren, die parallel ausgeführt werden.

Komplexe Stream-Verarbeitung

Sie können gerichtete azyklische Graphen (DAGs) von Kinesis Data Streams Streams-Anwendungen und Datenströmen erstellen. Dazu gehört in der Regel das Übertragen von Daten aus mehreren Anwendungen von Kinesis Data Streams in einen anderen Stream für die nachgeordnete Verarbeitung durch eine andere Anwendung von Kinesis Data Streams.

Vorteile der Verwendung von Kinesis Data Streams

Sie können mit Kinesis Data Streams eine Vielzahl von Problemen lösen, die beim Daten-Streaming auftreten. Typisch ist jedoch der Einsatz der Echtzeit-Aggregation von Daten gefolgt vom Laden der aggregierten Daten in ein Data Warehouse oder einen MapReduce-Cluster.

Die Daten werden in Kinesis-Daten-Streams geschrieben. Dies sorgt für Beständigkeit und Elastizität. Die Verzögerung zwischen dem Zeitpunkt, zu dem ein Datensatz in den Stream aufgenommen wird, und dem Zeitpunkt, zu dem er abgerufen werden kann (put-to-get Verzögerung), beträgt in der Regel weniger als 1 Sekunde. Eine Anwendung von Kinesis Data Streams kann somit die Daten aus dem Stream nahezu sofort nach dem Hinzufügen der Daten nutzen. Die von Kinesis Data Streams verwalteten Serviceaufgaben befreien Sie von dem Aufwand, der beim Erstellen und Ausführen einer Datenzufuhr-Pipeline betrieben werden muss. Sie können Streaming-Anwendungen des Typs Map-

Reduce erstellen. Die Elastizität von Kinesis Data Streams erlaubt das Skalieren des Streams, damit Sie niemals Datensätze verlieren, bevor diese ablaufen.

Mehrere Anwendungen von Kinesis Data Streams können Daten aus einem Stream aufnehmen, sodass mehrere Aktionen, beispielsweise Archivierung und Verarbeitung, gleichzeitig und unabhängig voneinander durchgeführt werden können. So ist es unter anderem möglich, dass zwei Anwendungen Daten aus demselben Stream auslesen. Die erste Anwendung berechnet die Ausführung von Aggregaten und aktualisiert eine Amazon-DynamoDB-Tabelle und die zweite Anwendung komprimiert und archiviert Daten in einem Datenspeicher wie Amazon Simple Storage Service (Amazon S3). Die DynamoDB-Tabelle mit laufenden Aggregaten wird dann von einem Dashboard für Berichte gelesen. up-to-the-minute

Die Kinesis Client Library ermöglicht eine fehlertolerante Nutzung von Daten aus Datenströmen und stellt zudem skalierbaren Support für Anwendungen von Kinesis Data Streams bereit.

Zugehörige Services

Weitere Informationen zur Verwendung von Amazon-EMR-Clustern zum direkten Lesen und Verarbeiten von Kinesis-Datenströmen finden Sie unter Kinesis Connector.

Terminologie und Konzepte von Amazon Kinesis Data Streams

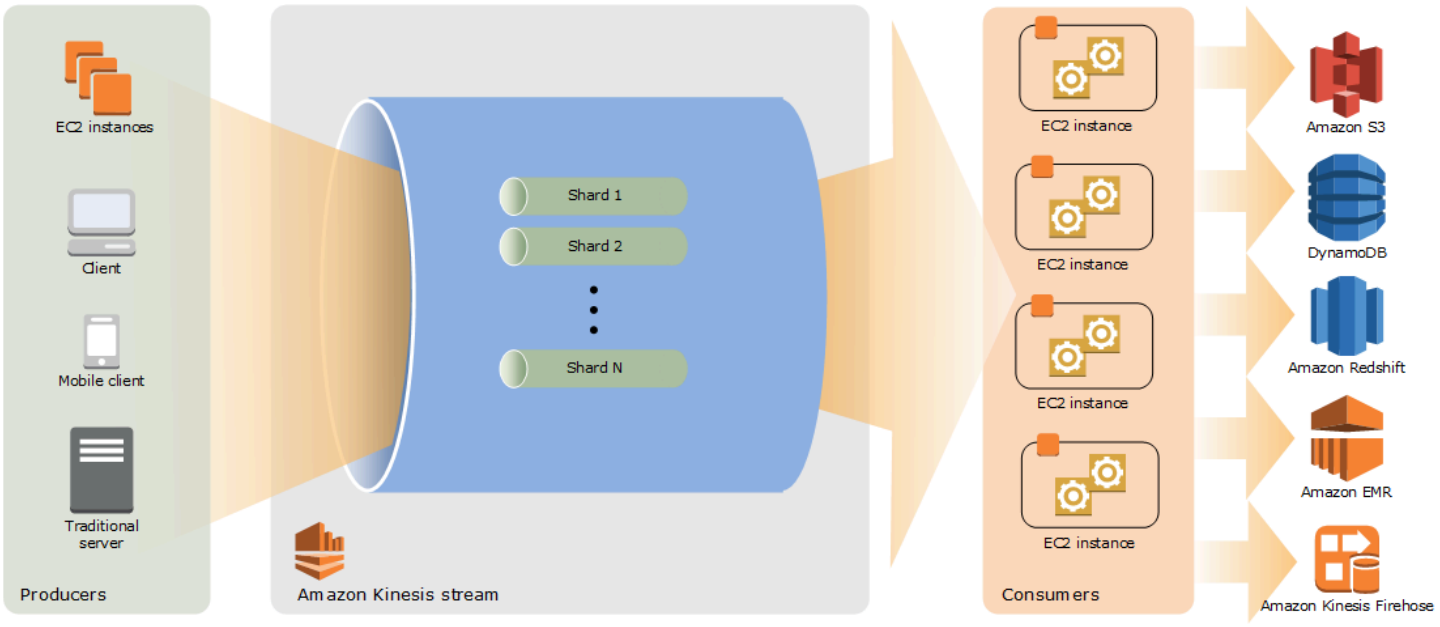
Bevor Sie mit Amazon Kinesis Data Streams beginnen, sollten Sie sich mit dessen Architektur und Terminologie vertraut machen.

Themen

- Informieren Sie sich über die High-Level-Architektur von Kinesis Data Streams
- Machen Sie sich mit der Terminologie von Kinesis Data Streams vertraut

Informieren Sie sich über die High-Level-Architektur von Kinesis Data Streams

Das folgende Diagramm zeigt die High-Level-Architektur von Kinesis Data Streams. Die Produzenten übermitteln kontinuierlich Daten an Kinesis Data Streams und die Verbraucher verarbeiten die Daten in Echtzeit. Verbraucher (z. B. eine benutzerdefinierte Anwendung, die auf Amazon EC2 läuft, oder ein Amazon Firehose Firehose-Lieferstream) können ihre Ergebnisse mit einem AWS Service wie Amazon DynamoDB, Amazon Redshift oder Amazon S3 speichern.



Machen Sie sich mit der Terminologie von Kinesis Data Streams vertraut

Kinesis Data Stream

Ein Kinesis-Datenstrom ist eine Gruppe von Shards. Jeder Shard hat eine Sequenz von Datensätzen. Jedem Datensatz wird von Kinesis Data Streams eine Sequenznummer zugewiesen.

Datensatz

Die von Kinesis Data Streams gespeicherte Dateneinheit wird als Datensatz bezeichnet. Datensätze besehen aus einer Sequenznummer, einem Partitionsschlüssel und einem Daten-Blob. Ein Daten-Blob ist eine unveränderliche Byte-Reihenfolge. Kinesis Data Streams untersucht, interpretiert oder ändert Daten im Blob in keiner Weise. Ein Daten-Blob kann bis zu 1 MB groß sein.

Kapazitätsmodus

Ein Datenstrom-Kapazitätsmodus bestimmt, wie Kapazität verwaltet wird und wie Ihnen die Nutzung Ihres Datenstroms in Rechnung gestellt wird. Derzeit können Sie in Kinesis Data Streams zwischen einem On-Demand-Modus und einem Bereitstellungsmodus für Ihre Datenstreams wählen. Weitere Informationen finden Sie unter Wählen Sie den richtigen Modus zum Streamen.

Im On-Demand-Modus verwaltet Kinesis Data Streams die Shards automatisch, um den erforderlichen Durchsatz bereitzustellen. Sie zahlen nur für den tatsächlich genutzten Durchsatz und Kinesis Data Streams passt sich automatisch dem Durchsatzbedarf Ihrer Workloads an, wenn dieser steigt oder sinkt. Weitere Informationen finden Sie unter Funktionen und Anwendungsfälle im On-Demand-Standardmodus.

Mit einem bereitgestellten Modus müssen Sie die Anzahl der Shards für den Datenstrom angeben. Die Gesamtkapazität eines Datenstroms ist die Summe der Kapazitäten der einzelnen Shards. Sie können die Anzahl der Shards in einem Datenstrom nach Bedarf erhöhen oder verringern, und die Anzahl der Shards wird Ihnen nach einem Stundensatz in Rechnung gestellt. Weitere Informationen finden Sie unter Funktionen und Anwendungsfälle im Bereitstellungsmodus.

Aufbewahrungszeitraum

Der Aufbewahrungszeitraum gibt den Zeitraum am, in dem auf Datensätze zugegriffen werden kann, nachdem sie dem Stream hinzugefügt wurden. Die Standard-Aufbewahrungsdauer eines

Streams beträgt nach Erstellung 24 Stunden. Sie können den Aufbewahrungszeitraum mithilfe des IncreaseStreamRetentionPeriodVorgangs auf bis zu 8760 Stunden (365 Tage) erhöhen und den Aufbewahrungszeitraum mithilfe des Vorgangs auf mindestens 24 Stunden reduzieren. DecreaseStreamRetentionPeriod Für Streams mit einem Aufbewahrungszeitraum von mehr als 24 Stunden fallen zusätzliche Gebühren an. Weitere Informationen finden Sie unter Preise für Amazon Kinesis Daten-Streams.

Produzent

Produzenten speichern Datensätze in Amazon Kinesis Data Streams. Ein Webserver, der Protokolldaten an einen Stream sendet, ist ein Beispiel für einen Produzenten.

Konsument

Verbraucher erhalten Datensätze von Amazon Kinesis Data Streams und verarbeiten sie. Diese Konsumenten werden als Anwendung von Amazon Kinesis Data Streams bezeichnet.

Anwendung von Amazon Kinesis Data Streams

Eine Anwendung von Amazon Kinesis Data Streams ist ein Verbraucher eines Streams, der in der Regel auf einer Flotte von EC2-Instances ausgeführt wird.

Sie können zwei Arten von Konsumenten entwickeln: geteilte Rundesendekonsumenten und erweiterte Rundesendekonsumenten. Informationen zu den Unterschieden zwischen diesen Typen und zu ihrer Erstellung erhalten Sie unter Daten aus Amazon Kinesis Data Streams lesen.

Die Ausgabe einer Anwendung von Kinesis Data Streams kann als Eingabe für einen anderen Stream genutzt werden. So können Sie komplexe Topologien erstellen, die Daten in Echtzeit verarbeiten. Eine Anwendung kann auch Daten an eine Vielzahl anderer AWS Dienste senden. Es sind mehrere Anwendungen für einen Stream möglich. Dabei kann jede Anwendung Daten aus einem Stream gleichzeitig und unabhängig konsumieren.

Shard

Ein Shard ist eine eindeutig identifizierbare Sequenz von Datensätzen in einem Stream. Ein Stream besteht aus einem oder mehreren Shards. Jeder Shard stellt eine feste Einheit der Kapazität bereit. Jeder Shard kann bis zu 5 Transaktionen pro Sekunde für Lesevorgänge unterstützen, bis zu einer maximalen Gesamtdatenleserate von 2 MB pro Sekunde und bis zu 1 000 Datensätze pro Sekunde für Schreibvorgänge, bis zu einer maximalen Gesamtdatenschreibrate von 1 MB pro Sekunde

(einschließlich Partitionsschlüssel). Die Datenkapazität Ihres Streams bezieht sich auf die Anzahl der Shards, die Sie für den Stream festlegen. Die Gesamtkapazität des Streams ist die Summe der Kapazitäten der einzelnen Shards.

Wenn sich Ihre Datenrate erhöht, können Sie die Anzahl der zu Ihrem Stream zugewiesenen Shards erhöhen oder verringern. Weitere Informationen finden Sie unter Einen Stream erneut teilen.

Partitionsschlüssel

Ein Partitionsschlüssel wird verwendet, um Daten nach Shard in einem Stream zu gruppieren. Kinesis Data Streams teilt die Datensätze eines Streams auf mehrere Shards auf. Dabei wird der Partitionsschlüssel verwendet, der jedem Datensatz zugeordnet ist, um zu ermitteln, zu welchem Shard ein gegebener Datensatz gehört. Partitionsschlüssel sind Unicode-Zeichenfolgen mit einer maximalen Länge von 256 Zeichen pro Schlüssel. Eine MD5 Hash-Funktion wird verwendet, um Partitionsschlüssel 128-Bit-Ganzzahlwerten zuzuordnen und zugehörige Datensätze unter Verwendung der Hash-Schlüsselbereiche der Shards Shards den Shards zuzuordnen. Wenn eine Anwendung Daten in einen Stream schreibt, muss sie einen Partitionsschlüssel angeben.

Sequenznummer

Jeder Datensatz verfügt über eine Sequenznummer, die pro Partitionsschlüssel im betreffenden Shard eindeutig ist. Die Sequenznummer wird von Kinesis Data Streams zugewiesen, nachdem Sie mit `client.putRecords` oder `client.putRecord` in den Stream geschrieben haben. Sequenznummern für denselben Partitionsschlüssel steigen in der Regel im Laufe der Zeit. Je länger die Zeitdauer zwischen Schreibanforderungen ist, desto größer wird die Sequenznummer.

> **Note**
>
> Sequenznummern können nicht als Index für Datensätze innerhalb desselben Streams verwendet werden. Nutzen Sie für die logische Unterteilung von Datensätzen Partitionsschlüssel oder erstellen Sie für jeden Datensatz einen separaten Stream.

Kinesis Client Library

Die Kinesis Client Library wird in Ihre Anwendung kompiliert, um eine fehlertolerante Nutzung von Daten aus dem Stream zu ermöglichen. Die Kinesis Client Library stellt sicher, dass für jeden Shard ein Datensatzprozessor läuft und diesen Shard verarbeitet. Die Bibliothek vereinfacht zudem das

Auslesen von Daten aus dem Stream. Die Kinesis Client Library verwendet Amazon DynamoDB-Tabellen zum Speichern von Metadaten im Zusammenhang mit dem Datenverbrauch. Sie erstellt drei Tabellen pro Anwendung, die Daten verarbeitet. Weitere Informationen finden Sie unter Verwenden Sie die Kinesis-Clientbibliothek.

Anwendungsname

Der Name einer Anwendung von Amazon Kinesis Data Streams identifiziert die Anwendung. Jede Ihrer Anwendungen muss einen eindeutigen Namen haben, der auf das AWS Konto und die Region zugeschnitten ist, die von der Anwendung verwendet werden. Dieser Name wird als Name für die Steuertabelle in Amazon DynamoDB und als Namespace für Amazon-Metriken verwendet. CloudWatch

Serverseitige Verschlüsselung

Amazon Kinesis Data Streams kann automatisch sensible Daten verschlüsseln, wenn ein Produzent diese an einen Stream übergibt. Kinesis Data Streams verwendet AWS KMS-Masterschlüssel für die Verschlüsselung. Weitere Informationen finden Sie unter Datenschutz in Amazon Kinesis Data Streams.

> **Note**
>
> Zum Auslesen aus einem verschlüsselten Stream oder zum Schreiben in einen solchen benötigen Konsumentenanwendungen eine Berechtigung für den Zugriff auf den Masterschlüssel. Informationen zum Gewähren von Berechtigungen für Produzenten- und Konsumentenanwendungen finden Sie unter the section called “Berechtigungen zur Verwendung von benutzergenerierten KMS-Schlüsseln”.

> **Note**
>
> Die Verwendung serverseitiger Verschlüsselung verursacht AWS Key Management Service () Kosten.AWS KMS Weitere Informationen finden Sie unter AWS Key Management Service – Preise.

Kontingente und -Einschränkungen

In der folgenden Tabelle werden Stream- und Shard-Kontingente und -Limits für Amazon Kinesis Data Streams beschrieben.

Kontingent	On-Demand-Modus	Modus bereitgestellter Kapazität
Anzahl der Datenströme	Es gibt kein oberes Kontingent für die Anzahl der Streams in Ihrem AWS Konto. Standardmäßig können Sie im On-Demand-Kapazitätsmodus bis zu 50 Datenströme erstellen. Wenn du eine Erhöhung dieses Kontingents benötigst, erstelle ein Support-Ticket.	Es gibt kein oberes Kontingent für die Anzahl der Streams im Bereitstellungsmodus innerhalb eines Kontos.
Anzahl der Shards	Es gibt keine Obergrenze. Die Anzahl der Shards hängt von der Menge der erfassten Daten und dem erforderlichen Durchsatz ab. Kinesis Data Streams skaliert die Anzahl der Shards automatisch als Reaktion auf Änderungen des Datenvolumens und des Datenverkehrs.	Es gibt keine Obergrenze. Das Standard-Shard-Kontingent beträgt 20.000 Shards pro AWS-Konto für Folgendes: AWS-Regionen • USA Ost (Nord-Virginia) • USA West (Oregon) • Europa (Irland) Für alle anderen Regionen beträgt das Standard-Shard-Kontingent 1.000 oder 6.000 Shards pro Stück. AWS-Konto Sie können das Shard-Kon

Kontingent	On-Demand-Modus	Modus bereitgestellter Kapazität
		tingent und die Nutzung Ihres Kontos in der Service Quotas Quotas-Konsole unter https://console.aws.amazon.com/servicequotas/einsehen. Um eine Erhöhung des Shard-Kontingents anzufordern, verwenden Sie die Service Quotas Quotas-Konsole oder AWS CLI. Weitere Informationen finden Sie unter Anfordern einer Kontingenterhöhung.

Kontingent	On-Demand-Modus	Modus bereitgestellter Kapazität
Datenstrom-Durchsatz	Standardmäßig haben neue Datenströme, die mit dem On-Demand-Kapazitätsmodus erstellt wurden, einen Schreibdurchsatz MB/s von 4 und einen Lesedurchsatz MB/s von 8% In USA Ost (Nord-Virginia), USA West (Oregon) und Europa (Irland) AWS-Regionen skalieren Datenströme mit dem On-Demand-Kapazitätsmodus auf bis zu 10% GB/s des Schreib- und 20% GB/s Lesedurchsatzes. In anderen Regionen skalieren Datenströme mit dem On-Demand-Kapazitätsmodus auf bis zu 200 MB/s Schreibdurchsätze und 400 MB/s Lesevorgänge. Wenn Sie für diese Regionen eine Erhöhung der Kapazität auf bis zu 10 GB/s Schreib- und 20 GB/s Lesekapazitäten benötigen, reichen Sie ein Support-Ticket ein.	Es gibt keine Obergrenze. Der maximale Durchsatz hängt von der Anzahl der für den Stream bereitgestellten Shards ab. Jeder Shard kann bis zu 1 MB/sec oder 1.000 records/sec Schreibdurchsätze oder bis zu 2 MB/sec oder 2.000 records/sec Lesedurchsätze unterstützen. Wenn Sie mehr Aufnahmekapazität benötigen, können Sie die Anzahl der Shards im Stream mithilfe der AWS-Managementkonsole oder der API problemlos erhöhen. UpdateShardCount
Daten-Nutzlastgröße	Die maximale Größe der Datennutzlast eines vorherigen Datensatzes `base64-encoding` beträgt bis zu 10 MiB. Kinesis ist für die Verarbeitung von intermittierenden großen Datensätzen (1—10 MiB groß) unter Verwendung von Burst-Kapazität konzipiert.	

Kontingent	On-Demand-Modus	Modus bereitgestellter Kapazität
`GetRecords` -Transaktionsgröße	GetRecordskann bis zu 10 MB Daten pro Anruf von einem einzigen Shard und bis zu 10.000 Datensätze pro Anruf abrufen. Jeder Aufruf von `GetRecords` gilt als eine Lese-Transaktion. Ein Shard kann bis zu fünf Lese-Transaktionen pro Sekunde unterstützen. Jede Lese-Transaktion kann bis zu 10.000 Datensätze mit einem Kontingent von 10 MB pro Transaktion liefern.	
Datenleserate pro Shard	Jeder Shard kann eine maximale Gesamtdatenleserate von bis zu 2 MB pro Sekunde über unterstützen. GetRecords Wenn ein Aufruf von `GetRecords` 10 MB zurückgibt, lösen nachfolgende Aufrufe innerhalb der nächsten 5 Sekunden eine Ausnahme aus.	
Anzahl der registrierten Verbraucher pro Datenstrom	Mit dem Kinesis On-Demand-Advantage-Modus können Sie bis zu 50 registrierte Verbraucher erstellen (Enhanced Fan-out). Mit den Modi Kinesis On-Demand Standard und Kinesis Provisioned können Sie bis zu 20 registrierte Verbraucher (Enhanced Fan-out Limit) für jeden Datenstream erstellen.	
Zwischen Bereitstellungs- und On-Demand-Modus wechseln	Für jeden Datenstream in Ihrem AWS Konto können Sie innerhalb von 24 Stunden zweimal zwischen den Modi „On-Demand-Kapazität“ und „Bereitgestellte Kapazität“ wechseln.	

Limits für API

Wie die meisten AWS APIs Operationen der Kinesis Data Streams API sind ratenbegrenzt. Die folgenden Beschränkungen gelten pro AWS Konto und Region. Weitere Informationen zu Kinesis Data Streams APIs finden Sie in der Amazon Kinesis API-Referenz.

KDS-Limits für die API der Steuerebene

Im folgenden Abschnitt werden die Grenzwerte für die KDS-Steuerungsebene beschrieben. APIs Mit der KDS-Steuerebene APIs können Sie Ihre Datenströme erstellen und verwalten. Diese Beschränkungen gelten pro AWS Konto und Region.

Limits für die API der Steuerebene

API	API-Aufruf-Limit	Pro Konto/Stream	Description
AddTagsToStream	5 Transaktionen pro Sekunde (TPS)	Pro Konto	50 Tags pro Datenstrom
CreateStream	5 TPS	Pro Konto	Es gibt kein Kontingent für die Anzahl der Streams in einem Konto. Sie erhalten beim Erstellen einer `CreateStream` -Anforderung eine `LimitExceededException` , wenn Sie versuchen , eine der folgenden Aktionen auszuführen: • Es befinden sich zu einem beliebigen Zeitpunkt mehr als fünf Streams im `CREATING`-Zustand. • Sie erstellen mehr Shards als für Ihr Konto zulässig sind.
DecreaseStreamRetentionPeriod	5 TPS	Pro Stream	Der Mindestwert für den Aufbewahrungszeitraum eines Datenstroms beträgt 24 Stunden.

API	API-Aufruf-Limit	Pro Konto/Stream	Description
DeleteResourcePolicy	5 TPS	Pro Konto	Wenn Sie eine Erhöhung dieses Limits benötigen, erstellen Sie bitte ein Support-Ticket.
DeleteStream	5 TPS	Pro Konto	
DeregisterStreamConsumer	5 TPS	Pro Stream	
DescribeAccountSettings	5 TPS	Pro Konto	
DescribeLimits	1 TPS	Pro Konto	
DescribeStream	10 TPS	Pro Konto	
DescribeStreamConsumer	20 TPS	Pro Stream	
DescribeStreamSummary	20 TPS	Pro Konto	
DisableEnhancedMonitoring	5 TPS	Pro Stream	
EnableEnhancedMonitoring	5 TPS	Pro Stream	
GetResourcePolicy	5 TPS	Pro Konto	Wenn Sie eine Erhöhung dieses Limits benötigen, erstellen Sie bitte ein Support-Ticket.

API	API-Aufruf-Limit	Pro Konto/Stream	Description
IncreaseStreamRetentionPeriod	5 TPS	Pro Stream	Der maximale Wert für den Aufbewahrungszeitraum eines Streams beträgt 8760 Stunden (365 Tage).
ListShards	1000 TPS	Pro Stream	
ListStreamConsumers	5 TPS	Pro Stream	
ListStreams	5 TPS	Pro Konto	
ListTagsForStream	5 TPS	Pro Stream	
MergeShards	5 TPS	Pro Stream	Gilt nur für bereitgestellte.
PutResourcePolicy	5 TPS	Pro Konto	Wenn Sie eine Erhöhung dieses Limits benötigen, erstellen Sie bitte ein Support-Ticket.

API	API-Aufruf-Limit	Pro Konto/Stream	Description
RegisterStreamConsumer	5 TPS	Pro Stream	Sie können pro Datenstrom bis zu 20 Verbraucher registrieren. Ein bestimmter Verbraucher kann jeweils nur für einen Datenstrom registriert werden. Es können nur 5 Verbraucher gleichzeitig erstellt werden. Mit anderen Worten, es können nicht mehr als 5 Verbraucher gleichzeitig den Status `CREATING` haben.
RemoveTagsFromStream	5 TPS	Pro Stream	
SplitShard	5 TPS	Pro Stream	Gilt nur für bereitgestellte
StartStreamEncryption		Pro Stream	Sie können innerhalb von 24 Stunden 25 Mal erfolgreich einen neuen AWS KMS-Schlüssel für die serverseitige Verschlüsselung anwenden.

API	API-Aufruf-Limit	Pro Konto/Stream	Description
StopStreamEncryption		Pro Stream	Sie können die serverseitige Verschlüsselung in einem fortlaufenden 24-Stunden-Zeitraum 25 Mal erfolgreich deaktivieren.
UpdateShardCount		Pro Stream	Gilt nur für bereitgestellte. Das Standardlimit für die Anzahl der Shards liegt bei 10.000. Es gibt zusätzliche Beschränkungen für diese API. Weitere Informationen finden Sie unter UpdateShardCount.
UpdateStreamMode		Pro Stream	Für jeden Datenstream in Ihrem AWS Konto können Sie innerhalb von 24 Stunden zweimal zwischen den Modi „On-Demand-Kapazität“ und „Bereitgestellte Kapazität“ wechseln.

API	API-Aufruf-Limit	Pro Konto/Stream	Description
UpdateStr eamWarmThroughput	5 TPS	Pro Konto	Der maximale Warmdurchsatz, der konfiguriert werden kann, entspricht dem Datenstrom-Durchsatzlimit des On-Demand-Modus für das Konto und die Region.
UpdateAccountSetti ngs	5 TPS	Pro Konto	Aktivieren oder deaktivieren Sie Kontoeinstellungen, z. B. den On-Demand-Advantage-Modus.

Limits für die KDS-API der Datenebene

Im folgenden Abschnitt werden die Grenzwerte für die KDS-Datenebene APIs beschrieben. Mit der APIs KDS-Datenebene können Sie Ihre Datenströme für die Erfassung und Verarbeitung von Datensätzen in Echtzeit verwenden. Diese Limits gelten pro Shard innerhalb Ihrer Datenströme.

Grenzwerte für die API der Datenebene

API	API-Aufruf-Limit	Nutzlast-Limit	Zusätzliche Einzelhei ten
GetRecords	5 TPS	Die maximale Anzahl von Datensätzen, die pro Anruf zurückgeg eben werden können, beträgt 10.000. Die maximale Größe der Daten, die `GetRecords`	Wenn ein Aufruf diese Datenmeng e zurückgibt, lösen nachfolgende Aufrufe innerhalb der nächsten 5 Sekunden eine `Provision edThrough`

API	API-Aufruf-Limit	Nutzlast-Limit	Zusätzliche Einzelheiten
		zurückgeben kann, ist 10 MB.	`putExceededException` aus. Wenn der bereitgestellte Durchsatz im Stream nicht ausreicht, werden nachfolgende Aufrufe innerhalb der nächsten Sekunde ausgeführt. `ProvisionedThroughputExceededException`
GetShardIterator	5 TPS		Ein Shard-Iterator läuft 5 Minuten nach Rückgabe an den Anforderer ab. Wenn eine GetShardIterator Anfrage zu oft gestellt wird, erhalten Sie eine ProvisionedThroughputExceededException.

API	API-Aufruf-Limit	Nutzlast-Limit	Zusätzliche Einzelheiten
PutRecord	1000 TPS	Jeder Shard kann Schreibvorgänge von bis zu 1.000 Datensätzen pro Sekunde unterstützen, was einer maximalen Datenschreibgeschwindigkeit von insgesamt 10 MiB pro Sekunde entspricht.	Kinesis ist für die Verarbeitung von intermittierenden großen Datensätzen (1—10 MiB groß) unter Verwendung von Burst-Kapazität konzipiert.
PutRecords		Jede PutRecords Anfrage kann bis zu 500 Datensätze unterstützen. Jeder Datensatz in der Anfrage kann bis zu 10 MiB groß sein, bis zu einer Grenze von 10 MiB für die gesamte Anfrage, einschließlich Partitionsschlüssel. Jeder Shard unterstützt Schreibvorgänge von bis zu 1.000 Datensätzen pro Sekunde bis zu einem maximalen Datenschreibvolumen von 1 MB pro Sekunde.	Kinesis ist für die Verarbeitung von intermittierenden großen Datensätzen (1—10 MiB groß) unter Verwendung von Burst-Kapazität konzipiert.

API	API-Aufruf-Limit	Nutzlast-Limit	Zusätzliche Einzelheiten
SubscribeToShard	Sie können SubscribeToShard pro Sekunde pro registriertem Benutzer und Shard einen Anruf tätigen.		Wenn Sie SubscribeToShard erneut mit demselben ConsumerARN anrufen und ShardId innerhalb von 5 Sekunden nach einem erfolgreichen Anruf erhalten Sie eineResourceInUseException.

Erhöhung der Kontingente

Sie können mit Service Quotas eine Erhöhung für ein Kontingent beantragen, sofern das Kontingent anpassbar ist. Einige Anfragen werden automatisch gelöst, während andere an den AWS Support weitergeleitet werden. Sie können den Status einer Anfrage zur Erhöhung des Kontingents verfolgen, die an den AWS Support gesendet wurde. Anfragen zur Erhöhung der Service Quotas werden nicht vorrangig unterstützt. Wenn Sie eine dringende Anfrage haben, wenden Sie sich an den AWS Support. Weitere Informationen finden Sie unter Was sind Service Quotas?.

Um eine Erhöhung des Servicekontingents anzufordern, befolgen Sie das unter Kontingenterhöhung beantragen beschriebenen Verfahren.

Vollständige Voraussetzungen für die Einrichtung von Amazon Kinesis Data Streams

Bevor Sie Amazon Kinesis Data Streams zum ersten Mal verwenden, führen Sie die folgenden Aufgaben durch, um Ihre Umgebung einzurichten.

Aufgaben

- Melden Sie sich an für AWS
- Bibliotheken und Tools herunterladen
- Konfigurieren Sie Ihre Entwicklungsumgebung

Melden Sie sich an für AWS

Wenn Sie sich für Amazon Web Services (AWS) registrieren, wird Ihr AWS Konto automatisch für alle Dienste angemeldet AWS, einschließlich Kinesis Data Streams. Berechnet werden Ihnen aber nur die Services, die Sie nutzen.

Wenn Sie bereits ein AWS Konto haben, fahren Sie mit der nächsten Aufgabe fort. Wenn Sie kein AWS -Konto haben, führen Sie die folgenden Schritte zum Erstellen eines Kontos aus.

Um sich für ein AWS Konto zu registrieren

1. Öffnen Sie https://portal.aws.amazon.com/billing/die Anmeldung.
2. Folgen Sie den Online-Anweisungen.

   Während der Anmeldung erhalten Sie einen Telefonanruf oder eine Textnachricht und müssen einen Verifizierungscode über die Telefontasten eingeben.

   Wenn Sie sich für eine anmelden AWS-Konto, Root-Benutzer des AWS-Kontoswird eine erstellt. Der Root-Benutzer hat Zugriff auf alle AWS-Services und Ressourcen des Kontos. Als bewährte Sicherheitsmethode weisen Sie einem Benutzer Administratorzugriff zu und verwenden Sie nur den Root-Benutzer, um Aufgaben auszuführen, die Root-Benutzerzugriff erfordern.

Bibliotheken und Tools herunterladen

Die folgenden Bibliotheken und Tools unterstützen Sie bei der Arbeit mit Kinesis Data Streams:

- Die Amazon-Kinesis-API-Referenz ist der grundlegende Satz von Vorgängen, die Kinesis Data Streams unterstützt. Weitere Informationen zum Durchführen grundlegender Operationen mit Java-Code finden Sie in den folgenden Ressourcen:
  - Entwickeln Sie Produzenten mithilfe der Amazon Kinesis Data Streams Streams-API mit dem AWS SDK für Java
  - Entwickeln Sie Verbraucher mit dem AWS SDK für Java
  - Kinesis-Datenstreams erstellen und verwalten
- Die AWS SDKs for Go, Java, .NET JavaScript, PHP, Python und Ruby beinhalten Kinesis Data Streams Streams-Unterstützung und Beispiele. Wenn Ihre Version von AWS SDK für Java keine Beispiele für Kinesis Data Streams enthält, können Sie sie auch von GitHubherunterladen.
- Die Kinesis Client Library (KCL) bietet ein easy-to-use Programmiermodell für die Datenverarbeitung. Die KCL unterstützt Sie beim schnellen Start mit Kinesis Data Streams in Java, Node.js, .NET, Python und Ruby. Weitere Informationen finden Sie unter Lesen von Daten aus Streams.
- Das AWS Command Line Interface unterstützt Kinesis Data Streams. Das AWS CLI ermöglicht es Ihnen, mehrere AWS Dienste von der Befehlszeile aus zu steuern und sie mithilfe von Skripten zu automatisieren.

Konfigurieren Sie Ihre Entwicklungsumgebung

Damit Sie die KCL verwenden können, müssen Sie sicherstellen, dass Ihre Java-Entwicklungsumgebung die folgenden Anforderungen erfüllt:

- Java 1.7 (Java SE 7 JDK) oder höher. Sie können die neueste Java-Software von der Oracle-Website unter Java SE-Downloads herunterladen.
- Apache Commons-Paket (Code, HTTP-Client und Protokollierung)
- Jackson JSON-Prozessor

Beachten Sie, dass das AWS SDK für Java Apache Commons und Jackson im Drittanbieter-Ordner enthält. Allerdings arbeitet das SDK for Java mit Java 1.6, während die Kinesis Client Library Java 1.7 erfordert.

Verwenden Sie die AWS CLI , um Amazon Kinesis Data Streams Streams-Operationen durchzuführen

In diesem Abschnitt erfahren Sie, wie Sie grundlegende Amazon Kinesis Data Streams Streams-Operationen mit dem AWS Command Line Interface durchführen. Sie erlernen Grundlagen für den Datenfluss von Kinesis Data Streams und die erforderlichen Schritte zum Senden von Daten an einen Kinesis-Datenstrom sowie zum Abrufen von Daten von dort.

Wenn Sie Kinesis Data Streams zum ersten Mal verwenden, sollten Sie sich zunächst mit den Konzepten und der Terminologie in Terminologie und Konzepte von Amazon Kinesis Data Streams vertraut machen.

Themen

- Tutorial: Installation und Konfiguration von AWS CLI for Kinesis Data Streams
- Tutorial: Führen Sie grundlegende Kinesis Data Streams Streams-Operationen mit dem AWS CLI

Für CLI-Zugriff benötigen Sie eine Zugriffsschlüssel-ID und einen geheimen Zugriffsschlüssel. Verwenden Sie möglichst temporäre Anmeldeinformationen anstelle langfristiger Zugriffsschlüssel. Temporäre Anmeldeinformationen bestehen aus einer Zugriffsschlüssel-ID, einem geheimen Zugriffsschlüssel und einem Sicherheits-Token, das angibt, wann die Anmeldeinformationen ablaufen. Weitere Informationen finden Sie unter Verwenden temporärer Anmeldeinformationen mit AWS Ressourcen im IAM-Benutzerhandbuch.

Eine ausführliche Anleitung zur Einrichtung von step-by-step IAM und Sicherheitsschlüsseln finden Sie unter Einen IAM-Benutzer erstellen.

In diesem Abschnitt werden die behandelten spezifischen Befehle unverändert übernommen, es sei denn, bestimmte Werte sind für jede Ausführung unbedingt unterschiedlich. Außerdem verwenden die Beispiele die USA West (Oregon)-Region, aber die Schritte in diesem Abschnitt funktionieren in allen Regionen, in denen Kinesis Data Streams unterstützt wird.

Tutorial: Installation und Konfiguration von AWS CLI for Kinesis Data Streams

Installieren Sie das AWS CLI

Ausführliche Schritte zur Installation der Betriebssysteme AWS CLI für Windows und Linux, OS X und Unix finden Sie unter AWS CLI installieren.

Verwenden Sie den folgenden Befehl, um die verfügbaren Optionen und Services anzuzeigen:

```
aws help
```

Sie werden den Kinesis Data Streams-Dienst verwenden, sodass Sie die AWS CLI Unterbefehle für Kinesis Data Streams mit dem folgenden Befehl überprüfen können:

```
aws kinesis help
```

Mit diesem Befehl erhalten Sie eine Ausgabe mit den verfügbaren Befehlen für Kinesis Data Streams:

```
AVAILABLE COMMANDS

       o add-tags-to-stream

       o create-stream

       o delete-stream

       o describe-stream

       o get-records

       o get-shard-iterator

       o help

       o list-streams

       o list-tags-for-stream

       o merge-shards
```

```
        o put-record

        o put-records

        o remove-tags-from-stream

        o split-shard

        o wait
```

Diese Befehlsliste entspricht der API von Kinesis Data Streams, die in der Amazon Kinesis Service API Reference beschrieben wird. Zum Beispiel entspricht der Befehl `create-stream`der API-Aktion `CreateStream`.

Der AWS CLI ist jetzt erfolgreich installiert, aber nicht konfiguriert. Dies wird im nächsten Abschnitt veranschaulicht.

Konfigurieren Sie das AWS CLI

Für den allgemeinen Gebrauch ist der `aws configure` Befehl der schnellste Weg, Ihre AWS CLI Installation einzurichten. Weitere Informationen finden Sie unter Konfiguration der AWS CLI.

Tutorial: Führen Sie grundlegende Kinesis Data Streams Streams-Operationen mit dem AWS CLI

In diesem Abschnitt wird die grundlegende Verwendung eines Kinesis-Datenstroms über die Befehlszeile AWS CLI beschrieben. Stellen Sie sicher, dass Sie mit den unter Terminologie und Konzepte von Amazon Kinesis Data Streams behandelten Konzepten vertraut sind.

> **Note**
>
> Nachdem Sie einen Stream erstellt haben, fallen für Ihr Konto geringe Gebühren für die Nutzung von Kinesis Data Streams an, da Kinesis Data Streams nicht für das AWS kostenlose Kontingent in Frage kommt. Wenn Sie mit diesem Tutorial fertig sind, löschen Sie Ihre AWS Ressourcen, damit keine Gebühren mehr anfallen. Weitere Informationen finden Sie unter Schritt 4: Bereinigen.

Themen

- Schritt 1: Erstellen Sie einen Stream
- Schritt 2: Einen Datensatz erstellen
- Schritt 3: Holen Sie sich den Datensatz
- Schritt 4: Bereinigen

Schritt 1: Erstellen Sie einen Stream

Der erste Schritt besteht darin, einen Stream zu erstellen und sicherzustellen, dass dieser erfolgreich erstellt wurde. Verwenden Sie den folgenden Befehl zum Erstellen eines Streams mit dem Namen „Foo”.

```
aws kinesis create-stream --stream-name Foo
```

Als Nächstes führen Sie den folgenden Befehl aus, um die Erstellung des Streams zu überprüfen:

```
aws kinesis describe-stream-summary --stream-name Foo
```

Sie sollten eine Ausgabe ähnlich dem folgenden Beispiel erhalten:

```
{
    "StreamDescriptionSummary": {
        "StreamName": "Foo",
        "StreamARN": "arn:aws:kinesis:us-west-2:123456789012:stream/Foo",
        "StreamStatus": "CREATING",
        "RetentionPeriodHours": 48,
        "StreamCreationTimestamp": 1572297168.0,
        "EnhancedMonitoring": [
            {
                "ShardLevelMetrics": []
            }
        ],
        "EncryptionType": "NONE",
        "OpenShardCount": 3,
        "ConsumerCount": 0
    }
}
```

In diesem Beispiel hat der Stream den Status CREATING, was bedeutet, dass er noch nicht einsatzbereit ist. Prüfen Sie den Status nach wenigen Minuten erneut. Jetzt sollten Sie eine Ausgabe ähnlich dem folgenden Beispiel erhalten:

```
{
    "StreamDescriptionSummary": {
        "StreamName": "Foo",
        "StreamARN": "arn:aws:kinesis:us-west-2:123456789012:stream/Foo",
        "StreamStatus": "ACTIVE",
        "RetentionPeriodHours": 48,
        "StreamCreationTimestamp": 1572297168.0,
        "EnhancedMonitoring": [
            {
                "ShardLevelMetrics": []
            }
        ],
        "EncryptionType": "NONE",
        "OpenShardCount": 3,
        "ConsumerCount": 0
    }
}
```

Diese Ausgabe enthält Informationen, die Sie für dieses Tutorial nicht benötigen. Die wichtigsten Informationen für den Moment sind`"StreamStatus": "ACTIVE"`, dass Sie wissen, dass der Stream bereit ist, verwendet zu werden, und die Informationen zu dem einzelnen Shard, den Sie angefordert haben. Sie können die Existenz Ihres neuen Streams auch mit dem Befehl `list-streams` überprüfen, wie in der folgenden Abbildung dargestellt:

```
aws kinesis list-streams
```

Ausgabe:

```
{
    "StreamNames": [
        "Foo"
    ]
}
```

Schritt 2: Einen Datensatz erstellen

Jetzt, da Sie über einen aktiven Stream verfügen, können Sie Daten an diesen senden. In diesem Tutorial verwenden Sie einen möglichst einfachen Befehl, `put-record`. Dieser sendet einen einzelnen Datensatz mit dem Text „testdata” in den Stream.

```
aws kinesis put-record --stream-name Foo --partition-key 123 --data testdata
```

Ist der Befehl erfolgreich, wird eine Ausgabe zurückgegeben, die wie folgt aussehen sollte:

```
{
    "ShardId": "shardId-000000000000",
    "SequenceNumber": "49546986683135544286507457936321625675700192471156785154"
}
```

Herzlichen Glückwunsch, Sie haben gerade Daten zu einem Stream hinzugefügt! Als Nächstes erfahren Sie, wie Sie Daten aus dem Stream abrufen können.

Schritt 3: Holen Sie sich den Datensatz

GetShardIterator

Bevor Sie Daten aus dem Stream abrufen können, müssen Sie den Shard-Iterator für den Shard abrufen, an dem Sie interessiert sind. Ein Shard-Iterator stellt die Position des Streams und des Shards dar, aus denen der Konsument (in diesem Fall der Befehl `get-record`) Daten ausliest. Sie verwenden den `get-shard-iterator` Befehl wie folgt:

```
aws kinesis get-shard-iterator --shard-id shardId-000000000000 --shard-iterator-type
 TRIM_HORIZON --stream-name Foo
```

Wie Sie wissen, gibt es eine API von Kinesis Data Streams zu den `aws kinesis`-Befehlen. Wenn Sie also mehr zu den angezeigten Parametern erfahren möchten, informieren Sie sich im API-Verweisthema `GetShardIterator`. Eine erfolgreiche Ausführung führt zu einer Ausgabe, die dem folgenden Beispiel ähnelt:

```
{
    "ShardIterator": "AAAAAAAAAAHSywljv0zEgPX4NyKdZ5wryMzP9yALs8NeKbUjp1IxtZs1Sp
+KEd9I6AJ9ZG4lNR1EMi+9Md/nHvtLyxpfhEzYvkTZ4D9DQVz/mBYWRO6OTZRKnW9gd
+efGN2aHFdkH1rJl4BL9Wyrk+ghYG22D2T1Da2EyNSH1+LAbK33gQweTJADBdyMwlo5r6PqcP2dzhg="
```

```
}
```

Die lange Zeichenfolge scheinbar zufälliger Zeichen ist der Shard-Iterator (Ihrer lautet jedoch anders). Sie müssen copy/paste den Shard-Iterator in den Befehl get eingeben, wie im Folgenden gezeigt. Shard-Iteratoren haben eine gültige Lebensdauer von 300 Sekunden. Dies sollte ausreichend Zeit sein, damit Sie copy/paste vom Shard-Iterator zum nächsten Befehl übergehen können. Sie müssen alle Zeilenumbrüche aus Ihrem Shard-Iterator entfernen, bevor Sie sie in den nächsten Befehl einfügen. Wenn Sie eine Fehlermeldung erhalten, dass der Shard-Iterator nicht mehr gültig ist, führen Sie den Befehl erneut aus. `get-shard-iterator`

GetRecords

Der Befehl `get-records` ruft Daten aus dem Stream ab und wird in einen Aufruf an `GetRecords` in der API von Kinesis Data Streams aufgelöst. Der Shard-Iterator gibt die Position im Shard an, ab der Datensätze sequenziell ausgelesen werden sollen: Wenn keine Datensätze in dem Teil des Shards verfügbar sind, auf die der Iterator zeigt, gibt `GetRecords` eine leere Liste zurück. Möglicherweise sind mehrere Aufrufe erforderlich, um zu einem Teil des Shards zu gelangen, der Datensätze enthält.

Im folgenden Beispiel für den `get-records` Befehl:

```
aws kinesis get-records --shard-iterator
 AAAAAAAAAAHSywljv0zEgPX4NyKdZ5wryMzP9yALs8NeKbUjp1IxtZs1Sp+KEd9I6AJ9ZG4lNR1EMi
+9Md/nHvtLyxpfhEzYvkTZ4D9DQVz/mBYWRO6OTZRKnW9gd+efGN2aHFdkH1rJl4BL9Wyrk
+ghYG22D2T1Da2EyNSH1+LAbK33gQweTJADBdyMwlo5r6PqcP2dzhg=
```

Wenn Sie dieses Tutorial von einem UNIX-Befehlsprozessor wie der Bash aus ausführen, können Sie die Erfassung des Shard-Iterators mit einem verschachtelten Befehl wie dem Folgenden automatisieren:

```
SHARD_ITERATOR=$(aws kinesis get-shard-iterator --shard-id shardId-000000000000 --
shard-iterator-type TRIM_HORIZON --stream-name Foo --query 'ShardIterator')

aws kinesis get-records --shard-iterator $SHARD_ITERATOR
```

Wenn Sie dieses Tutorial von einem System aus ausführen, das dies unterstützt PowerShell, können Sie die Erfassung des Shard-Iterators mit einem Befehl wie dem folgenden automatisieren:

```
aws kinesis get-records --shard-iterator ((aws kinesis get-shard-iterator --shard-id
 shardId-000000000000 --shard-iterator-type TRIM_HORIZON --stream-name Foo).split('"')
[4])
```

Das erfolgreiche Ergebnis des `get-records` Befehls fordert Datensätze aus Ihrem Stream für den Shard an, den Sie beim Abrufen des Shard-Iterators angegeben haben, wie im folgenden Beispiel:

```
{
  "Records":[ {
    "Data":"dGVzdGRhdGE=",
    "PartitionKey":"123”,
    "ApproximateArrivalTimestamp": 1.441215410867E9,
    "SequenceNumber":"495449852569073700275708858640655777030226526385964318 74"
  } ],
  "MillisBehindLatest":24000,

 "NextShardIterator":"AAAAAAAAAAEDOW3ugseWPE4503kqN1yN1UaodY8unE0sYslMUmC6lX9hlig5+t4RtZM0/
tALfiI4QGjunVgJvQsjxjh2aLyxaAaPr
+LaoENQ7eVs4EdYXgKyThTZGPcca2fVXYJWL3yafv9dsDwsYVedI66dbMZFC8rPMWc797zxQkv4pSKvPOZvrUIudb8UkH3V
}
```

Beachten Sie, dass `get-records` dies oben als Anfrage beschrieben wurde, was bedeutet, dass Sie möglicherweise keine oder mehr Datensätze erhalten, selbst wenn Ihr Stream Datensätze enthält. Alle zurückgegebenen Datensätze stellen möglicherweise nicht alle Datensätze dar, die sich derzeit in Ihrem Stream befinden. Das ist normal, und der Produktionscode fragt den Stream in angemessenen Intervallen nach Datensätzen ab. Diese Abfragegeschwindigkeit hängt von Ihren spezifischen Anforderungen an das Anwendungsdesign ab.

In Ihren Aufzeichnungen in diesem Teil des Tutorials werden Sie feststellen, dass die Daten unbrauchbar zu sein scheinen — und es ist nicht der Klartext, den `testdata` wir gesendet haben. Dies liegt an der Art und Weise, auf die `put-record` Base64-Codierung verwendet, um Ihnen das Senden von Binärdaten zu ermöglichen. Die Kinesis Data Streams Streams-Unterstützung in bietet jedoch AWS CLI keine Base64-Decodierung, da die Base64-Decodierung von rohen Binärinhalten, die auf stdout gedruckt werden, auf bestimmten Plattformen und Terminals zu unerwünschtem Verhalten und potenziellen Sicherheitsproblemen führen kann. Wenn Sie einen Base64-Decoder verwenden (z. B. https://www.base64decode.org/), um `dGVzdGRhdGE=` manuell zu decodieren, sehen Sie, dass es tatsächlich `testdata` ist. Dies ist für dieses Tutorial ausreichend, da der in der Praxis nur selten zur Nutzung von Daten verwendet wird. AWS CLI Häufiger wird es verwendet, um den Status des Streams zu überwachen und Informationen abzurufen, wie zuvor gezeigt (`describe-stream`und`list-streams`). Weitere Informationen zur KCL finden Sie unter Entwickeln benutzerdefinierter Verbraucher mit gemeinsamem Durchsatz mithilfe von KCL.

`get-records`gibt nicht immer alle Datensätze im angegebenen Stream/Shard zurück. Verwenden Sie in diesem Fall den `NextShardIterator` aus dem letzten Ergebnis, um die nächste

Datensatzgruppe abzurufen. Wenn mehr Daten in den Stream eingegeben würden, was bei Produktionsanwendungen der Normalfall ist, könnten Sie jedes Mal nach nutzbaren Daten suchen. `get-records` Wenn Sie jedoch innerhalb der Lebensdauer von 300 Sekunden nicht `get-records` mit dem nächsten Shard-Iterator aufrufen, erhalten Sie eine Fehlermeldung, und Sie müssen den `get-shard-iterator` Befehl verwenden, um einen neuen Shard-Iterator zu erhalten.

In dieser Ausgabe wird ebenfalls `MillisBehindLatest` bereitgestellt. Dabei handelt es sich um die Anzahl der Millisekunden, die die Antwort des GetRecords-Vorgangs vom vorderen Ende des Streams entfernt ist. Dies zeigt an, wie weit der Konsument hinter der aktuellen Zeit zurückliegt. Der Wert Null gibt an, dass die Datenverarbeitung aktuell ist und dass zurzeit keine neuen zu verarbeitenden Datensätze vorhanden sind. In diesem Tutorial sehen Sie möglicherweise eine recht große Zeit, wenn Sie sich die Zeit dafür genommen haben, stets mitzulesen. Standardmäßig verbleiben Datensätze 24 Stunden in einem Stream und warten darauf, dass Sie sie abrufen. Dieser Zeitraum wird als Aufbewahrungszeitraum bezeichnet und ist bis zu 365 Tage konfigurierbar.

Ein erfolgreiches `get-records` Ergebnis wird immer angezeigt, `NextShardIterator` auch wenn sich derzeit keine weiteren Datensätze im Stream befinden. Hierbei handelt es sich um ein Abrufmodell, bei dem davon ausgegangen wird, dass ein Produzent zu einem bestimmten Zeitpunkt möglicherweise mehr Datensätze in den Stream einfügt. Auch wenn Sie Ihre eigene Abrufroutine schreiben können, wird Ihnen das Abrufen abgenommen, wenn Sie die zuvor erwähnte KCL zum Entwickeln von Konsumentenanwendungen verwenden.

Wenn Sie `get-records` so lange aufrufen, bis der Stream und der Shard, aus dem Sie abrufen, keine Datensätze mehr enthalten, erhalten Sie eine Ausgabe mit leeren Datensätzen, die dem folgenden Beispiel ähnelt:

```
{
    "Records": [],
    "NextShardIterator": "AAAAAAAAAAGCJ5jzQNjmdhO6B/YDIDE56jmZmrmMA/r1WjoHXC/
kPJXc1rckt3TFL55dENfe5meNgdkyCRpUPGzJpMgYHaJ53C3nCAjQ6s7ZupjXeJGoUFs5oCuFwhP+Wul/
EhyNeSs5DYXLSSC5XCapmCAYGFjYER69QSdQjxMmBPE/hiybFDi5qtkT6/PsZNz6kFoqtDk="
}
```

Schritt 4: Bereinigen

Lösche deinen Stream, um Ressourcen freizugeben und ungewollte Gebühren für dein Konto zu vermeiden. Tun Sie dies jedes Mal, wenn Sie einen Stream erstellt haben und ihn nicht verwenden werden, da Gebühren pro Stream anfallen, unabhängig davon, ob Sie Daten mit ihm übertragen und abrufen oder nicht. Der Befehl zum Aufräumen lautet wie folgt:

```
aws kinesis delete-stream --stream-name Foo
```

Erfolg führt zu keiner Ausgabe. Verwenden Sie`describe-stream`, um den Fortschritt des Löschvorgangs zu überprüfen:

```
aws kinesis describe-stream-summary --stream-name Foo
```

Wenn Sie diesen Befehl unmittelbar nach dem Löschbefehl ausführen, erhalten Sie eine Ausgabe, die dem folgenden Beispiel ähnelt:

```
{
    "StreamDescriptionSummary": {
        "StreamName": "samplestream",
        "StreamARN": "arn:aws:kinesis:us-west-2:123456789012:stream/samplestream",
        "StreamStatus": "ACTIVE",
```

Nachdem der Stream vollständig gelöscht wurde, führt `describe-stream` zum Fehler „nicht gefunden”.

```
A client error (ResourceNotFoundException) occurred when calling the
 DescribeStreamSummary operation:
Stream Foo under account 123456789012 not found.
```

Tutorials „Erste Schritte“ für Amazon Kinesis Data Streams

Amazon Kinesis Data Streams bietet eine Reihe verschiedener Lösungen für die Aufnahme und Nutzung von Daten aus Kinesis-Datenströmen. Die Tutorials in diesem Abschnitt sollen Ihnen helfen, die Konzepte und Funktionen von Amazon Kinesis Data Streams besser zu verstehen und die Lösung zu finden, die Ihren Anforderungen entspricht.

Themen

- Tutorial: Verarbeiten Sie Aktiendaten in Echtzeit mit KPL und KCL 2.x
- Tutorial: Verarbeiten Sie Aktiendaten in Echtzeit mit KPL und KCL 1.x
- Tutorial: Analysieren Sie Aktiendaten in Echtzeit mit Amazon Managed Service für Apache Flink
- Tutorial: Verwendung AWS Lambda mit Amazon Kinesis Data Streams
- Verwenden Sie die AWS Streaming-Datenlösung für Amazon Kinesis

Tutorial: Verarbeiten Sie Aktiendaten in Echtzeit mit KPL und KCL 2.x

Das Szenario für dieses Tutorial beinhaltet die Aufnahme von Aktiengeschäften in einen Datenstream und das Schreiben einer grundlegenden Amazon Kinesis Data Streams Streams-Anwendung, die Berechnungen im Stream durchführt. Sie lernen, wie Sie einen Stream von Datensätzen an Kinesis Data Streams senden und eine Anwendung implementieren, die die Datensätze nahezu in Echtzeit verarbeitet und verarbeitet.

> **Important**
>
> Nachdem Sie einen Stream erstellt haben, fallen für Ihr Konto geringe Gebühren für die Nutzung von Kinesis Data Streams an, da Kinesis Data Streams nicht für das AWS kostenlose Kontingent in Frage kommt. Nach dem Start der Konsumentenanwendung fallen nominale Gebühren für die Amazon-DynamoDB-Nutzung an. Die Konsumentenanwendung verwendet DynamoDB zum Verfolgen des Verarbeitungsstatus. Wenn Sie mit dieser Anwendung fertig sind, sollten Sie Ihre AWS -Ressourcen löschen, damit keine weiteren Gebühren anfallen. Weitere Informationen finden Sie unter Bereinigen von Ressourcen.

Der Code greift nicht auf tatsächliche Wertpapierdaten zu, sondern simuliert nur deren Strom. Dazu werden zufällige Wertpapierdaten erzeugt. Ausgangspunkt sind dabei echte Marktdaten der 25 führenden Aktien gemäß Börsenkapitalisierung von Februar 2015. Wenn Sie Zugriff auf einen Echtzeit-Stream von Wertpapierdaten haben, möchten Sie vermutlich nützliche, zeitnahe Statistiken aus den Stream-Daten erzeugen. Sie können beispielsweise eine Zeitfensteranalyse durchführen, um festzustellen, welche Aktie in den letzten 5 Minuten am häufigsten erworben wurde. Oder Sie möchten im Falle eines zu großen Verkaufsauftrags (d. h. zu viele Anteile) benachrichtigt werden. Der Code in diesem Tutorial kann erweitert werden, um solche Funktionen bereitzustellen.

Sie können die in diesem Tutorial aufgeführten Schritte mit einem Desktop-PC oder Laptop durchführen und sowohl den Produzenten- als auch den Verbrauchercode auf demselben Rechner oder auf jeder Plattform ausführen, die den definierten Anforderungen entspricht.

Bei den gezeigten Beispielen wird die Region USA West (Oregon) verwendet. Sie funktionieren aber auch für alle anderen AWS -Regionen, die Kinesis Data Streams unterstützen.

Aufgaben

- Erfüllen der Voraussetzungen
- Erstellen Sie einen Datenstream
- Erstellen Sie eine IAM-Richtlinie und einen Benutzer
- Laden Sie den Code herunter und erstellen Sie ihn
- Implementieren Sie den Produzenten
- Implementieren Sie den Verbraucher
- (Optional) Erweitern Sie den Consumer
- Bereinigen von Ressourcen

Erfüllen der Voraussetzungen

Sie müssen die folgenden Voraussetzungen erfüllen, um dieses Tutorial abschließen zu können:

Erstellen und verwenden Sie ein Amazon Web Services Services-Konto

Bevor Sie beginnen, stellen Sie sicher, dass Sie mit den unter besprochenen Konzepten vertraut sindTerminologie und Konzepte von Amazon Kinesis Data Streams, insbesondere mit Streams, Shards, Produzenten und Verbrauchern. Es ist zudem sinnvoll, die Schritte in der folgenden Anleitung durchzuführen: Tutorial: Installation und Konfiguration von AWS CLI for Kinesis Data Streams.

Sie benötigen ein AWS Konto und einen Webbrowser, um auf die AWS-Managementkonsole zugreifen zu können.

Verwenden Sie für den Konsolenzugriff Ihren IAM-Benutzernamen und Ihr Passwort, um sich über die IAM-Anmeldeseite bei AWS-Managementkonsole anzumelden. Informationen zu AWS Sicherheitsanmeldedaten, einschließlich programmatischem Zugriff und Alternativen zu langfristigen Anmeldeinformationen, finden Sie unter AWS Sicherheitsanmeldedaten im IAM-Benutzerhandbuch. Einzelheiten zur Anmeldung bei Ihrem AWS-Konto finden Sie unter So melden Sie sich an AWS im AWS-Anmeldung Benutzerhandbuch.

Weitere Informationen zu IAM und dem Einrichten von Sicherheitsschlüsseln finden Sie unter Erstellen eines IAM-Benutzers.

Erfüllen Sie die Anforderungen an die Systemsoftware

Im System, das Sie zum Ausführen der Anwendung verwenden, muss Java 7 oder höher installiert sein. Zum Herunterladen und Installieren des neuesten Java Development Kit (JDK) rufen Sie die Website für die Java SE-Installation von Oracle auf.

Sie benötigen die neueste AWS SDK für Java-Version.

Die Consumer-Anwendung benötigt die Kinesis Client Library (KCL) Version 2.2.9 oder höher, die Sie unter /tree/master abrufen GitHub können. https://github.com/awslabs/ amazon-kinesis-client

Nächste Schritte

Erstellen Sie einen Datenstream

Erstellen Sie einen Datenstream

Zuerst müssen Sie den Daten-Stream erstellen, den Sie in den weiteren Schritten dieses Tutorials verwenden.

So erstellen Sie einen Stream

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Kinesis-Konsole unter https://console.aws.amazon.com/kinesis.
2. Klicken Sie im Navigationsbereich auf Data Streams (Daten-Streams).
3. Erweitern Sie in der Navigationsleiste die Regionsauswahl und wählen Sie eine Region aus.

4. Wählen Sie Create Kinesis Stream (Kinesis-Stream erstellen).
5. Geben Sie einen Namen für den Daten-Stream ein (z. B. **`StockTradeStream`**).
6. Geben Sie **1** die Anzahl der Shards ein, lassen Sie aber Estimate the number of shards you'll need (Schätzung der Anzahl der Shards, die Sie benötigen) reduziert.
7. Wählen Sie Create Kinesis Stream (Kinesis-Stream erstellen).

Auf der Listenseite Kinesis Streams (Kinesis-Streams) lautet der Status des Streams `CREATING`, während der Stream erstellt wird. Sobald der Stream verwendet werden kann, ändert sich der Status in `ACTIVE`.

Wenn Sie den Namen des Streams auswählen, wird auf der angezeigten Seite auf der Registerkarte Details eine Zusammenfassung der Konfiguration des Daten-Streams angezeigt. Der Abschnitt Monitoring (Überwachung) zeigt Überwachungsinformationen für den Stream an.

Nächste Schritte

Erstellen Sie eine IAM-Richtlinie und einen Benutzer

Erstellen Sie eine IAM-Richtlinie und einen Benutzer

Bewährte Sicherheitsmethoden für AWS schreiben die Verwendung detaillierter Berechtigungen vor, um den Zugriff auf verschiedene Ressourcen zu kontrollieren. AWS Identity and Access Management (IAM) ermöglicht Ihnen die Verwaltung von Benutzern und Benutzerberechtigungen in. AWS Eine IAM-Richtlinie führt explizit zulässige Aktionen sowie Ressourcen auf, für die diese Aktionen relevant sind.

Im Folgenden werden die Berechtigungen für Produzenten und Konsumenten von Kinesis Data Streams aufgelistet, die mindestens erforderlich sind.

Produzent

Aktionen	Ressource	Zweck
`DescribeStream` , `DescribeStreamSummary` , `DescribeStreamConsumer`	Kinesis Data Stream	Vor dem Versuch, Datensätze zu lesen, prüft der Verbrauch eam vorhanden und ob er aktiv ist und ob Shards im Daten

Aktionen	Ressource	Zweck
`SubscribeToShard` , `RegisterStreamConsumer`	Kinesis Data Stream	Abonniert und registriert Verbraucher bei einem Shard.
`PutRecord` , `PutRecords`	Kinesis Data Stream	Schreibt Datensätze in Kinesis Data Streams.

Konsument

Aktionen	Ressource	Zweck
`DescribeStream`	Kinesis Data Stream	Vor dem Versuch, Datensätze zu lesen, prüft der Verbrauch eam vorhanden und ob er aktiv ist und ob Shards im Daten
`GetRecords` , `GetShardIterator`	Kinesis Data Stream	Auslesen von Datensätzen aus einem Shard.
`CreateTable` , `DescribeTable` , `GetItem`, `PutItem`, `Scan`, `UpdateItem`	Amazon-DynamoDB-Tabelle.	Wenn der Verbraucher mit der Kinesis Client Library (KCL) entwickelt wird, benötigt er Berechtigungen für die Dynamo Verarbeitungsstatus der Anwendung zu verfolgen.
`DeleteItem`	Amazon-DynamoDB-Tabelle.	Für den Fall, dass der Verbraucher split/merge Operationen Streams Streams-Shards ausführt.
`PutMetricData`	CloudWatch Amazon-Protokoll	Die KCL lädt auch Metriken hoch CloudWatch, die für die Ü Anwendung nützlich sind.

Für dieses Tutorial erstellen Sie eine einzelne IAM-Richtlinie, die alle oben genannten Berechtigungen gewährt. Im Praxiseinsatz können Sie zwei Richtlinien erstellen, eine für Produzenten und eine für Verbraucher.

So erstellen Sie eine IAM-Richtlinie

1. Suchen Sie den Amazon-Ressourcennamen (ARN) für den neuen Datenstream, den Sie im vorherigen Schritt erstellt haben. Sie finden diesen ARN als Stream ARN (Stream-ARN) oben auf der Registerkarte Details. Das ARN-Format sieht folgendermaßen aus:

   ```
   arn:aws:kinesis:region:account:stream/name
   ```

   Region

   Der AWS Regionalcode, zum Beispiel`us-west-2`. Weitere Informationen finden Sie unter Regionen und Verfügbarkeitskonzepte.

   Konto

   Die AWS Konto-ID, wie in den Kontoeinstellungen angezeigt.

   Name

   Der Name des Datenstroms, den Sie im vorherigen Schritt erstellt haben, nämlich`StockTradeStream`.

2. Bestimmen Sie den ARN für die DynamoDB-Tabelle, die vom Verbraucher verwendet und von der ersten Verbraucher-Instance erstellt wird. Er muss das folgende Format aufweisen:

   ```
   arn:aws:dynamodb:region:account:table/name
   ```

   Die Region und die Konto-ID sind identisch mit den Werten im ARN des Datenstroms, den Sie für dieses Tutorial verwenden, aber der Name ist der Name der DynamoDB-Tabelle, die von der Verbraucheranwendung erstellt und verwendet wird. KCL verwendet den Anwendungsnamen als Tabellennamen. Verwenden Sie `StockTradesProcessor` in diesem Schritt für den DynamoDB-Tabellennamen, da dies der Anwendungsname ist, der in späteren Schritten in diesem Tutorial verwendet wird.

3. Wählen Sie in der IAM-Konsole unter Policies (https://console.aws.amazon.com/iam/home#policies) die Option Create policy aus. Wenn Sie zum ersten Mal mit IAM-Richtlinien arbeiten, wählen Sie Erste Schritte, Richtlinie erstellen aus.
4. Wählen Sie Select (Auswählen) neben Policy Generator (Richtliniengenerator) aus.
5. Wählen Sie Amazon Kinesis als AWS Service.

6. Legen Sie `DescribeStream`, `GetShardIterator`, `GetRecords`, `PutRecord` und `PutRecords` als zulässige Aktionen fest.
7. Geben Sie den ARN des Daten-Streams ein, den Sie in diesem Tutorial verwenden.
8. Verwenden Sie Add Statement (Statement hinzufügen) für die folgenden Elemente:

AWS Dienst	Aktionen	ARN
Amazon DynamoDB	`CreateTable` , `DeleteItem` , `DescribeTable` , `GetItem`, `PutItem`, `Scan`, `UpdateItem`	Der ARN der DynamoDB-Tabelle, den Sie in Schritt 2 dieses Verfahrens erstellt haben.
Amazon CloudWatch	`PutMetricData`	*

   Das Sternchen (*), das bei der Angabe einer ARN verwendet wird, ist nicht erforderlich. In diesem Fall liegt das daran, dass es keine bestimmte Ressource gibt, für die die `PutMetricData` Aktion aufgerufen wird. CloudWatch
9. Wählen Sie Next Step (Weiter) aus.
10. Ändern Sie Policy Name (Richtlinienname) in `StockTradeStreamPolicy`, prüfen Sie den Code und wählen sie Create Policy (Richtlinie erstellen) aus.

Das resultierende Richtliniendokument sollte folgendermaßen aussehen:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt123",
            "Effect": "Allow",
            "Action": [
                "kinesis:DescribeStream",
                "kinesis:PutRecord",
                "kinesis:PutRecords",
```

```
                "kinesis:GetShardIterator",
                "kinesis:GetRecords",
                "kinesis:ListShards",
                "kinesis:DescribeStreamSummary",
                "kinesis:RegisterStreamConsumer"
            ],
            "Resource": [
                "arn:aws:kinesis:us-west-2:111122223333:stream/StockTradeStream"
            ]
        },
        {
            "Sid": "Stmt234",
            "Effect": "Allow",
            "Action": [
                "kinesis:SubscribeToShard",
                "kinesis:DescribeStreamConsumer"
            ],
            "Resource": [
                "arn:aws:kinesis:us-west-2:111122223333:stream/StockTradeStream/
*"
            ]
        },
        {
            "Sid": "Stmt456",
            "Effect": "Allow",
            "Action": [
                "dynamodb:*"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-west-2:111122223333:table/
StockTradesProcessor"
            ]
        },
        {
            "Sid": "Stmt789",
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricData"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
```

```
}
```

So erstellen Sie einen IAM-Benutzer

1. Öffnen Sie unter https://console.aws.amazon.com/iam/ die IAM-Konsole.
2. Wählen Sie auf der Seite Users (Benutzer) die Option Add user (Benutzer hinzufügen) aus.
3. Geben Sie `StockTradeStreamUser` als Benutzername ein.
4. Wählen Sie für Access type (Zugriffstyp) die Option Programmatic access (Programmgesteuerter Zugriff) und wählen Sie dann Next: Permissions (Weiter: Berechtigungen).
5. Wählen Sie Vorhandene Richtlinien direkt zuordnen.
6. Suchen Sie nach dem Namen der Richtlinie, die Sie im vorherigen Verfahren erstellt haben (`StockTradeStreamPolicy`. Markieren Sie das Kontrollkästchen links neben dem Richtliniennamen und wählen Sie dann Next: Review (Weiter: Prüfen).
7. Überprüfen Sie die Details und die Zusammenfassung und wählen Sie dann Create user (Benutzer erstellen) aus.
8. Kopieren Sie die Access key ID (Zugriffsschlüssel-ID) und speichern Sie sie privat. Wählen Sie unter Secret access key (Geheimer Zugriffsschlüssel) die Option Show (Anzeigen) und speichern Sie auch diesen Schlüssel privat.
9. Fügen Sie die Zugriffs- und Geheimschlüssel in eine lokalen Datei an einem sichern Ort ein, auf den nur Sie Zugriff haben. Erstellen Sie für diese Anwendung eine Datei namens `~/.aws/credentials` (mit strikten Berechtigungen). Die Datei sollte das folgende Format aufweisen:

```
[default]
aws_access_key_id=access key
aws_secret_access_key=secret access key
```

So fügen Sie einem Benutzer eine IAM-Richtlinie hinzu

1. Klicken Sie in der IAM;-Konsole auf Richtlinien und wählen Sie Richtlinienaktionen aus.
2. Klicken Sie auf `StockTradeStreamPolicy` und Attach (Verknüpfen).
3. Wählen Sie `StockTradeStreamUser` und Attach Policy (Richtlinie anfügen) aus.

Nächste Schritte

Laden Sie den Code herunter und erstellen Sie ihn

Laden Sie den Code herunter und erstellen Sie ihn

In diesem Thema finden Sie Beispielcode zur Implementierung der Beispiel-Wertpapiertransaktionen im Daten-Stream (Produzent) und zur Verarbeitung dieser Daten (Verbraucher).

So laden Sie den Code herunter und erstellen ihn

1. Laden Sie den Quellcode aus dem https://github.com/aws-samples/amazon-kinesis-learning GitHub Repo auf Ihren Computer herunter.
2. Erstellen Sie in der IDE ein Projekt mit dem Quellcode. Behalten Sie die bereitgestellte Verzeichnisstruktur bei.
3. Fügen Sie dem Projekt die folgenden Bibliotheken hinzu:
   - Amazon Kinesis Client Library (KCL)
   - AWS SDK
   - Apache HttpCore
   - Apache HttpClient
   - Apache Commons Lang
   - Apache Commons Logging
   - Guava (Google-Kernbibliotheken für Java)
   - Jackson Annotations
   - Jackson Core
   - Jackson Databind
   - Jackson Dataformat: CBOR
   - Joda Time
4. Je nach IDE wird das Projekt möglicherweise automatisch erstellt. Wenn nicht, erstellen Sie es mit den für Ihre IDE erforderlichen Schritten.

Wenn Sie diese Schritte erfolgreich abgeschlossen haben, können Sie zum nächsten Abschnitt wechseln, the section called “Implementieren Sie den Produzenten”.

Nächste Schritte

Implementieren Sie den Produzenten

Dieses Tutorial verwendet das reale Szenario einer Überwachung des Wertpapierhandels. Die folgenden Prinzipien erläutern kurz, wie dieses Szenario zum Produzenten und seiner unterstützenden Codestruktur passt.

Beachten Sie den Quellcode und prüfen Sie die folgenden Informationen.

StockTrade Klasse

Ein bestimmter Wertpapierhandel wird durch eine Instance der StockTrade-Klasse dargestellt. Diese Instance enthält folgende Attribute: Tickersymbol, Preis, Anzahl der Anteile, Art des Handels (Kauf oder Verkauf) und ID zur eindeutigen Identifizierung der Handelsaktion. Dieser Klasse wird für Sie implementiert.

Stream-Datensatz

Ein Stream ist eine Sequenz von Datensätzen. Ein Datensatz ist die Serialisierung einer `StockTrade`-Instance im JSON-Format. Beispiel:

```
{
  "tickerSymbol": "AMZN",
  "tradeType": "BUY",
  "price": 395.87,
  "quantity": 16,
  "id": 3567129045
}
```

StockTradeGenerator Klasse

StockTradeGenerator hat eine Methode namens`getRandomTrade()`, die bei jedem Aufruf einen neuen zufällig generierten Aktienhandel zurückgibt. Dieser Klasse wird für Sie implementiert.

StockTradesWriter Klasse

Die `main`-Methode des Produzenten, StockTradesWriter, ruft kontinuierlich eine zufällige Handelsaktion ab und sendet die Daten an Kinesis Data Streams, indem sie die folgenden Aufgaben durchführt:

1. Liest den Datenstromnamen und den Regionsnamen als Eingabe.
2. Verwendet die`KinesisAsyncClientBuilder`, um die Region, die Anmeldeinformationen und die Client-Konfiguration festzulegen.
3. Sicherstellen, dass der Stream vorhanden und aktiv ist (wenn nicht, kommt es zu einer Beendigung mit Fehler).
4. Aufrufen der `StockTradeGenerator.getRandomTrade()`-Methode in einer Dauerschleife und anschließend Aufruf der `sendStockTrade`-Methode, um die Handelsdaten alle 100 Millisekunden an den Stream zu senden.

Die `sendStockTrade`-Methode der `StockTradesWriter`-Klasse hat den folgenden Code:

```
private static void sendStockTrade(StockTrade trade, KinesisAsyncClient
 kinesisClient,
            String streamName) {
        byte[] bytes = trade.toJsonAsBytes();
        // The bytes could be null if there is an issue with the JSON serialization
 by the Jackson JSON library.
        if (bytes == null) {
            LOG.warn("Could not get JSON bytes for stock trade");
            return;
        }

        LOG.info("Putting trade: " + trade.toString());
        PutRecordRequest request = PutRecordRequest.builder()
                .partitionKey(trade.getTickerSymbol()) // We use the ticker symbol
 as the partition key, explained in the Supplemental Information section below.
                .streamName(streamName)
                .data(SdkBytes.fromByteArray(bytes))
                .build();
        try {
            kinesisClient.putRecord(request).get();
        } catch (InterruptedException e) {
            LOG.info("Interrupted, assuming shutdown.");
        } catch (ExecutionException e) {
            LOG.error("Exception while sending data to Kinesis. Will try again next
 cycle.", e);
        }
    }
```

Beachten Sie die folgende Code-Struktur:

- Die `PutRecord` API erwartet ein Byte-Array, und Sie müssen den Handel in das JSON-Format konvertieren. Diese einzelne Codezeile führt die Operation aus:

```
byte[] bytes = trade.toJsonAsBytes();
```

- Bevor Sie die Transaktion senden können, erstellen Sie eine neue `PutRecordRequest`-Instance (in diesem Fall Anforderung genannt). Jede `request` benötigt den Namen des Streams, einen Partitionsschlüssel und einen Daten-Blob.

```
PutPutRecordRequest request = PutRecordRequest.builder()
    .partitionKey(trade.getTickerSymbol()) // We use the ticker symbol as the
 partition key, explained in the Supplemental Information section below.
    .streamName(streamName)
    .data(SdkBytes.fromByteArray(bytes))
    .build();
```

  Das Beispiel verwendet einen Börsenticker als Partitionsschlüssel, der den Datensatz einem bestimmten Shard zuordnet. In der Praxis sollten Sie Hunderte oder gar Tausende von Partitionsschlüsseln pro Shard haben, sodass die Datensätze in Ihrem Stream gleichmäßig verteilt sind. Weitere Informationen zum Hinzufügen von Daten zu einem Stream finden Sie unter Daten in Amazon Kinesis Data Streams schreiben.

  `request` kann die Daten jetzt an den Client senden (PUT-Operation):

```
   kinesisClient.putRecord(request).get();
```

- Eine Fehlerüberprüfung und Protokollierung sind immer nützliche Ergänzungen. Dieser Code protokolliert Fehlerbedingungen:

```
if (bytes == null) {
    LOG.warn("Could not get JSON bytes for stock trade");
    return;
```

```
}
```

Fügen Sie den try/catch Block rund um die `put` Operation hinzu:

```
try {
  kinesisClient.putRecord(request).get();
} catch (InterruptedException e) {
            LOG.info("Interrupted, assuming shutdown.");
} catch (ExecutionException e) {
            LOG.error("Exception while sending data to Kinesis. Will try again
 next cycle.", e);
}
```

Der Grund besteht darin, dass eine Kinesis Data Streams-PUT-Operation aufgrund eines Netzwerkfehlers fehlschlagen kann oder gedrosselt wird, weil die Durchsatzgrenze des Streams erreicht wird. Es wird empfohlen, dass Sie Ihre Wiederholungsrichtlinie für `put` Operationen sorgfältig prüfen, um Datenverlust zu vermeiden, z. B. die Verwendung eines Wiederholungsversuchs.

- Eine Statusprotokollierung ist hilfreich, wenn auch optional:

```
LOG.info("Putting trade: " + trade.toString());
```

Der hier gezeigte Produzent verwendet die API-Funktionalität von Kinesis Data Streams für einzelne Datensätze `PutRecord`. In der Praxis ist es oft effizienter, die Eignung von `PutRecords` für mehrere Datensätze zu nutzen und mehrere Datensatzstapel gleichzeitig zu senden, wenn ein Produzent viele Datensätze erstellt. Weitere Informationen finden Sie unter Daten in Amazon Kinesis Data Streams schreiben.

So führen Sie den Produzenten aus

1. Verifizieren Sie, dass der in Erstellen Sie eine IAM-Richtlinie und einen Benutzer abgerufene Zugriffsschlüssel samt geheimem Schlüsselpaar in der Datei `~/.aws/credentials` gespeichert wurde.
2. Führen Sie die `StockTradeWriter`-Klasse mit den folgenden Argumenten aus:

```
StockTradeStream us-west-2
```

Wenn Sie den Stream in einer anderen Region als `us-west-2` erstellt haben, müssen Sie stattdessen hier diese Region angeben.

Die Ausgabe sollte folgendermaßen oder ähnlich aussehen:

```
Feb 16, 2015 3:53:00 PM
com.amazonaws.services.kinesis.samples.stocktrades.writer.StockTradesWriter
 sendStockTrade
INFO: Putting trade: ID 8: SELL 996 shares of BUD for $124.18
Feb 16, 2015 3:53:00 PM
com.amazonaws.services.kinesis.samples.stocktrades.writer.StockTradesWriter
 sendStockTrade
INFO: Putting trade: ID 9: BUY 159 shares of GE for $20.85
Feb 16, 2015 3:53:01 PM
com.amazonaws.services.kinesis.samples.stocktrades.writer.StockTradesWriter
 sendStockTrade
INFO: Putting trade: ID 10: BUY 322 shares of WMT for $90.08
```

Ihre Wertpapierdaten werden nun von Kinesis Data Streams eingespeist.

Nächste Schritte

Implementieren Sie den Verbraucher

Implementieren Sie den Verbraucher

Die Verbraucheranwendung in diesem Tutorial verarbeitet die Wertpapiertransaktionen im Daten-Stream kontinuierlich. Sie gibt dann für jede Minute die beliebtesten Aktien aus, die gekauft und verkauft wurden. Die Anwendung setzt auf Kinesis Client Library (KCL) auf, die viele der mühsamen Arbeiten einer Verbraucheranwendung übernimmt. Weitere Informationen finden Sie unter Informationen zu KCL 1.x und 2.x.

Überprüfen Sie die folgenden Informationen in Bezug auf den Quellcode.

StockTradesProcessor Klasse

Die für Sie bereitgestellte Hauptklasse des Verbrauchers, die die folgenden Aufgaben erfüllt:

- Liest die als Argumente übergebenen Anwendungs-, Datenstrom- und Regionsnamen.
- Erzeugt eine `KinesisAsyncClient` Instanz mit dem Namen der Region.
- Erstellt eine `StockTradeRecordProcessorFactory`-Instance für die Instances von `ShardRecordProcessor`, implementiert von einer `StockTradeRecordProcessor`-Instance.
- Erzeugt eine `ConfigsBuilder` Instanz mit der `StockTradeRecordProcessorFactory` Instanz `KinesisAsyncClient` `StreamNameApplicationName`,, und. Dies ist für das Erstellen aller Konfigurationen mit Standardwerten nützlich.
- KCL-Scheduler (zuvor in den KCL-Versionen 1.x als KCL-Worker bezeichnet) mit der `ConfigsBuilder`-Instance erstellen.
- Der Scheduler erstellt für jeden Shard (der dieser Verbraucher-Instance zugeordnet ist) einen neuen Thread, der in einer Schleife die Datensätze aus dem Daten-Stream liest. Anschließend wird die `StockTradeRecordProcessor`-Instance aufgerufen, um die empfangenen Datensatzstapel zu verarbeiten.

StockTradeRecordProcessor Klasse

Implementierung der `StockTradeRecordProcessor`-Instance, die wiederum fünf erforderliche Methoden implementiert: `initialize`, `processRecords`, `leaseLost`, `shardEnded` und `shutdownRequested`.

Die Methoden `initialize` und `shutdownRequested` werden von KCL verwendet, um dem Datensatzverarbeiter mitzuteilen, wann er bereit sein muss, Datensätze zu empfangen, und wann der Empfang von Datensätzen gestoppt werden muss, damit alle anwendungsspezifischen Einrichtungs- und Beendigungsaufgaben ausgeführt werden können. `leaseLost` und `shardEnded` werden verwendet, um Logik zu implementieren, die beim Verlust eines Lease oder bei Erreichen des Shards-Endes im Rahmen der Shard-Verarbeitung auszuführen ist. In diesem Beispiel protokollieren wir einfach Meldungen dieser Ereignisse.

Der Code für diese Methoden wird für Sie bereitgestellt. Die wesentliche Verarbeitung erfolgt mit der `processRecords` Methode, die wiederum `processRecord` für die einzelnen Datensätze nutzt. Die letztgenannte Methode wird als nahezu leerer Skeleton-Code bereitgestellt und im nächsten Schritt (der weitere Informationen enthält) implementiert.

Beachten Sie außerdem die Implementierung der Hilfsmethoden für `processRecord`: `reportStats` und `resetStats`, die im ursprünglichen Quellcode leer sind.

Die `processRecords`-Methode wurde für Sie implementiert und führt die folgenden Schritte aus:

- Für jeden übergebenen Datensatz wird `processRecord` aufgerufen.
- Ruft `reportStats()` zum Drucken der neuesten Statistiken auf, wenn seit dem letzten Bericht mindestens 1 Minute vergangen ist, und dann `resetStats()`, um die Statistiken zu löschen, damit das nächste Intervall nur neue Datensätze enthält.
- Legt den Zeitpunkt für die nächste Berichterstellung fest.
- Ruft `checkpoint()` auf, wenn seit dem letzten Prüfpunkt mindestens 1 Minute vergangen ist.
- Legt den Zeitpunkt für das nächste Checkpointing fest.

Diese Methode verwendet für das Checkpointing und die Berichterstellung ein Intervall von 60 Sekunden. Weitere Informationen zum Checkpointing finden Sie unter Verwendung der Kinesis Client Library.

StockStats Klasse

Diese Klasse stellt eine Datenaufbewahrung und eine Nachverfolgung von Statistiken für die beliebtesten Aktien bereit. Dieser Code wird für Sie bereitgestellt und enthält folgende Methoden:

- `addStockTrade(StockTrade)`: fügt die angegebene `StockTrade` in die ausgeführten Statistiken ein.
- `toString()`: gibt die Statistiken als formatierte Zeichenfolge zurück.

Diese Klasse verfolgt die beliebtesten Aktien, indem sie fortlaufend die Gesamtzahl der Trades für jede Aktie und deren maximale Anzahl zählt. Sie aktualisiert diese Werte, sobald eine neue Handelstransaktion empfangen wird.

Fügen Sie Code zu den Methoden der `StockTradeRecordProcessor`-Klasse hinzu, wie in den folgenden Schritten gezeigt.

So implementieren Sie den Konsumenten

1. Implementieren Sie die `processRecord`-Methode, indem Sie ein richtig bemessenes `StockTrade`-Objekt instanziieren und die Datensatzdaten zu diesem hinzufügen, sodass im Falle eines Problems eine Warnung protokolliert wird.

```
byte[] arr = new byte[record.data().remaining()];
record.data().get(arr);
StockTrade trade = StockTrade.fromJsonAsBytes(arr);
    if (trade == null) {
        log.warn("Skipping record. Unable to parse record into StockTrade.
 Partition Key: " + record.partitionKey());
        return;
        }
stockStats.addStockTrade(trade);
```

2. Implementieren Sie eine `reportStats` Methode. Ändern Sie das Ausgabeformat nach Ihren Wünschen.

```
System.out.println("****** Shard " + kinesisShardId + " stats for last 1 minute
 ******\n" +
stockStats + "\n" +
"*****************************************************************\n");
```

3. Implementieren Sie die Methode `resetStats`, die eine neue `stockStats`-Instance erstellt.

```
stockStats = new StockStats();
```

4. Implementieren Sie die folgenden Methoden, die für die `ShardRecordProcessor` Schnittstelle erforderlich sind:

```
@Override
public void leaseLost(LeaseLostInput leaseLostInput) {
    log.info("Lost lease, so terminating.");
}

@Override
public void shardEnded(ShardEndedInput shardEndedInput) {
    try {
        log.info("Reached shard end checkpointing.");
        shardEndedInput.checkpointer().checkpoint();
    } catch (ShutdownException | InvalidStateException e) {
        log.error("Exception while checkpointing at shard end. Giving up.", e);
```

```
    }
}

@Override
public void shutdownRequested(ShutdownRequestedInput shutdownRequestedInput) {
    log.info("Scheduler is shutting down, checkpointing.");
    checkpoint(shutdownRequestedInput.checkpointer());
}

private void checkpoint(RecordProcessorCheckpointer checkpointer) {
    log.info("Checkpointing shard " + kinesisShardId);
    try {
        checkpointer.checkpoint();
    } catch (ShutdownException se) {
        // Ignore checkpoint if the processor instance has been shutdown (fail
 over).
        log.info("Caught shutdown exception, skipping checkpoint.", se);
    } catch (ThrottlingException e) {
        // Skip checkpoint when throttled. In practice, consider a backoff and
 retry policy.
        log.error("Caught throttling exception, skipping checkpoint.", e);
    } catch (InvalidStateException e) {
        // This indicates an issue with the DynamoDB table (check for table,
 provisioned IOPS).
        log.error("Cannot save checkpoint to the DynamoDB table used by the Amazon
 Kinesis Client Library.", e);
    }
}
```

So führen Sie den Konsumenten aus

1. Führen Sie den unter erstellten Produzenten aus, um simulierte Wertpapiertransaktionsdatensätze in den Stream zu schreiben.
2. Stellen Sie sicher, dass der Zugriffsschlüssel und das geheime Schlüsselpaar, die vorher (beim Erstellen des IAM-Benutzers) abgerufen wurden, in der Datei `~/.aws/credentials` gespeichert sind.
3. Führen Sie die `StockTradesProcessor`-Klasse mit den folgenden Argumenten aus:

```
StockTradesProcessor StockTradeStream us-west-2
```

Beachten Sie, dass Sie, wenn Sie Ihren Stream in einer anderen Region als `us-west-2` erstellt haben, stattdiesen diese Region hier angeben müssen.

Nach einer Minute sollen Sie eine Ausgabe ähnlich der folgenden sehen, die anschließend einmal pro Minute aktualisiert wird:

```
****** Shard shardId-000000000001 stats for last 1 minute ******
Most popular stock being bought: WMT, 27 buys.
Most popular stock being sold: PTR, 14 sells.
************************************************************
```

Nächste Schritte

[(Optional) Erweitern Sie den Consumer]

(Optional) Erweitern Sie den Consumer

Dieser optionale Abschnitt zeigt, wie Sie den Konsumentencode erweitern können, um einem komplexeren Szenario gerecht zu werden.

Wenn Sie minütlich über die größten Verkaufsaufträge informiert werden möchten, können Sie die `StockStats`-Klasse an drei Stellen bearbeiten.

So erweitern Sie den Konsumenten

1. Fügen Sie neue Instance-Variablen hinzu:

```
// Ticker symbol of the stock that had the largest quantity of shares sold
private String largestSellOrderStock;
// Quantity of shares for the largest sell order trade
private long largestSellOrderQuantity;
```

2. Fügen Sie folgenden Code zu hinz `addStockTrade`:

```
if (type == TradeType.SELL) {
     if (largestSellOrderStock == null || trade.getQuantity() >
 largestSellOrderQuantity) {
         largestSellOrderStock = trade.getTickerSymbol();
         largestSellOrderQuantity = trade.getQuantity();
     }
 }
```

3. Ändern Sie die `toString`-Methode, um die zusätzlichen Informationen zu drucken:

```
public String toString() {
    return String.format(
        "Most popular stock being bought: %s, %d buys.%n" +
        "Most popular stock being sold: %s, %d sells.%n" +
        "Largest sell order: %d shares of %s.",
        getMostPopularStock(TradeType.BUY),
 getMostPopularStockCount(TradeType.BUY),
        getMostPopularStock(TradeType.SELL),
 getMostPopularStockCount(TradeType.SELL),
        largestSellOrderQuantity, largestSellOrderStock);
}
```

Wenn Sie den Konsumenten jetzt ausführen (führen Sie auch den Produzenten), sollten Sie eine Ausgabe ähnlich der folgenden sehen:

```
****** Shard shardId-000000000001 stats for last 1 minute ******
Most popular stock being bought: WMT, 27 buys.
Most popular stock being sold: PTR, 14 sells.
Largest sell order: 996 shares of BUD.
************************************************************
```

Nächste Schritte

Bereinigen von Ressourcen

Bereinigen von Ressourcen

Da Sie für die Nutzung des Kinesis Data Streams zahlen, sollten Sie diesen ebenso wie die entsprechende Amazon-DynamoDB-Tabelle löschen, wenn sie nicht mehr benötigt werden. Für einen aktiven Stream fallen auch dann nominale Gebühren an, wenn Sie keine Datensätze senden oder abrufen. Der Grund hierfür ist, dass ein aktiver Stream durch kontinuierlichen „Horchen“ darauf, ob neue Datensätze oder Anforderungen zum Abrufen von Datensätzen eingehen, Ressourcen verbraucht.

So löschen Sie den Stream und die Tabelle

1. Schalten Sie alle Produzenten und Verbraucher aus, die Sie möglicherweise noch in Betrieb haben.
2. Öffnen Sie die Kinesis-Konsole unter https://console.aws.amazon.com/kinesis.
3. Wählen Sie den Stream aus, den Sie für diese Anwendung erstellt haben (`StockTradeStream`).
4. Wählen Sie Delete Stream (Stream löschen) aus.
5. Öffnen Sie die DynamoDB-Konsole unter. https://console.aws.amazon.com/dynamodb/
6. Löschen Sie die `StockTradesProcessor`-Tabelle.

Zusammenfassung

Die Verarbeitung einer großen Datenmenge nahezu in Echtzeit erfordert weder das Schreiben von kompliziertem Code noch die Entwicklung einer riesigen Infrastruktur. Es ist genauso einfach wie die Schreiblogik, eine kleine Datenmenge zu verarbeiten (wie Schreiben`processRecord(Record)`), aber Kinesis Data Streams zu verwenden, um so zu skalieren, dass es für eine große Menge an Streaming-Daten funktioniert. Sie müssen sich keine Gedanken über die Skalierung der Verarbeitung machen, da Kinesis Data Streams das für Sie übernimmt. Sie müssen lediglich Ihre Streaming-Datensätze an Kinesis Data Streams senden und eine Logik für die Verarbeitung neu empfangener Datensätze schreiben.

Nachfolgend einige mögliche Erweiterungen für diese Anwendung.

Shard-übergreifende Aggregation

Derzeit erhalten Sie Statistiken, die aus der Aggregation von Datensätzen resultieren, die von einem einzelnen Auftragnehmer eines einzelnen Shards empfangen werden. (Ein Shard

kann jeweils nur von einem Auftragnehmer in einer einzelnen Anwendung verarbeitet werden.) Möglicherweise möchten Sie, wenn Sie eine Skalierung durchführen und mehr als einen Shard haben, eine Shard-übergreifende Aggregation vornehmen. Dies kann mit einer Pipeline-Architektur realisiert werden, bei der die Ausgabe eines jeden Auftragnehmers in einen anderen Stream mit einem einzelnen Shard eingespeist wird, der von einem Auftragnehmer verarbeitet wird, der die Ausgaben der ersten Phase aggregiert. Da die Daten aus der ersten Phase begrenzt sind (eine Ausgabe pro Minute pro Shard), können sie problemlos von nur einem Shard verarbeitet werden.

Skalierung

Wenn der Stream skaliert wird, damit mehr Shards zur Verfügung stehen (da viele Produzenten Daten senden), geschieht dies dadurch, dass mehr Worker hinzugefügt werden. Sie können die Auftragnehmer in EC2-Instances ausführen und Auto-Scaling-Gruppen nutzen.

Verwenden Sie Konnektoren zu Amazon S3 S3/ DynamoDB/Amazon Redshift/Storm

Da ein Stream kontinuierlich verarbeitet wird, kann seine Ausgabe an andere Ziele gesendet werden. AWS bietet Konnektoren zur Integration von Kinesis Data Streams mit anderen AWS Diensten und Tools von Drittanbietern.

Tutorial: Verarbeiten Sie Aktiendaten in Echtzeit mit KPL und KCL 1.x

Im Szenario dieses Tutorials werden Wertpapierdaten in einen Datenstrom geschrieben. Zudem wird eine einfache Anwendung mit Amazon Kinesis Data Streams erstellt, die Berechnungen mit dem Stream durchführt. Sie lernen, wie Sie einen Stream von Datensätzen an Kinesis Data Streams senden und eine Anwendung implementieren, die die Datensätze nahezu in Echtzeit verarbeitet und verarbeitet.

Important

Nachdem Sie einen Stream erstellt haben, fallen für Ihr Konto geringe Gebühren für die Nutzung von Kinesis Data Streams an, da Kinesis Data Streams nicht für das AWS kostenlose Kontingent in Frage kommt. Nach dem Start der Konsumentenanwendung fallen nominale Gebühren für die Amazon-DynamoDB-Nutzung an. Die Konsumentenanwendung verwendet DynamoDB zum Verfolgen des Verarbeitungsstatus. Wenn Sie mit dieser

Anwendung fertig sind, sollten Sie Ihre AWS -Ressourcen löschen, damit keine weiteren Gebühren anfallen. Weitere Informationen finden Sie unter Bereinigen von Ressourcen.

Der Code greift nicht auf tatsächliche Wertpapierdaten zu, sondern simuliert nur deren Strom. Dazu werden zufällige Wertpapierdaten erzeugt. Ausgangspunkt sind dabei echte Marktdaten der 25 führenden Aktien gemäß Börsenkapitalisierung von Februar 2015. Wenn Sie Zugriff auf einen Echtzeit-Stream von Wertpapierdaten haben, möchten Sie vermutlich nützliche, zeitnahe Statistiken aus den Stream-Daten erzeugen. Sie können beispielsweise eine Zeitfensteranalyse durchführen, um festzustellen, welche Aktie in den letzten 5 Minuten am häufigsten erworben wurde. Oder Sie möchten im Falle eines zu großen Verkaufsauftrags (d. h. zu viele Anteile) benachrichtigt werden. Der Code in diesem Tutorial kann erweitert werden, um solche Funktionen bereitzustellen.

Sie können die in diesem Tutorial aufgeführten Schritte auf einem Desktop-PC oder Laptop durchführen und sowohl den Produzenten- als auch den Konsumentencode auf demselben Rechner oder einer Plattform ausführen, die die angegebenen Anforderungen erfüllt, beispielsweise Amazon Elastic Compute Cloud (Amazon EC2).

Bei den gezeigten Beispielen wird die Region USA West (Oregon) verwendet. Sie funktionieren aber auch für alle anderen AWS -Regionen, die Kinesis Data Streams unterstützen.

Aufgaben

- Erfüllen der Voraussetzungen
- Erstellen Sie einen Datenstream
- Erstellen Sie eine IAM-Richtlinie und einen Benutzer
- Laden Sie den Implementierungscode herunter und erstellen Sie ihn
- Implementieren Sie den Produzenten
- Implementieren Sie den Verbraucher
- (Optional) Erweitern Sie die Zahl der Verbraucher
- Bereinigen von Ressourcen

Erfüllen der Voraussetzungen

Für Tutorial: Verarbeiten Sie Aktiendaten in Echtzeit mit KPL und KCL 1.x gelten die folgenden Anforderungen.

Erstellen und verwenden Sie ein Amazon Web Services Services-Konto

Bevor Sie beginnen, müssen Sie sich mit den in Terminologie und Konzepte von Amazon Kinesis Data Streams behandelten Konzepten vertraut machen, insbesondere mit Streams, Shards, Produzenten und Konsumenten. Das Durcharbeiten von Tutorial: Installation und Konfiguration von AWS CLI for Kinesis Data Streams ist ebenfalls hilfreich.

Sie benötigen ein AWS Konto und einen Webbrowser, um auf das zugreifen zu können AWS-Managementkonsole.

Verwenden Sie für den Konsolenzugriff Ihren IAM-Benutzernamen und Ihr Passwort, um sich über die IAM-Anmeldeseite bei AWS-Managementkonsole anzumelden. Informationen zu AWS Sicherheitsanmeldedaten, einschließlich programmatischem Zugriff und Alternativen zu langfristigen Anmeldeinformationen, finden Sie unter AWS Sicherheitsanmeldedaten im IAM-Benutzerhandbuch. Einzelheiten zur Anmeldung bei Ihrem AWS-Konto finden Sie unter So melden Sie sich an AWS im AWS-Anmeldung Benutzerhandbuch.

Weitere Informationen zu IAM und dem Einrichten von Sicherheitsschlüsseln finden Sie unter Erstellen eines IAM-Benutzers.

Erfüllen Sie die Anforderungen an die Systemsoftware

Auf dem System, dass die Anwendung ausführt, muss Java 7 oder höher installiert sein. Zum Herunterladen und Installieren des neuesten Java Development Kit (JDK) rufen Sie die Website für die Java SE-Installation von Oracle auf.

Wenn Sie über eine Java IDE wie Eclipse verfügen, können Sie den Quellcode öffnen, bearbeiten, erstellen und ausführen.

Sie benötigen die neueste AWS SDK für Java-Version. Wenn Sie Eclipse als IDE (integrierte Entwicklungsumgebung) nutzen, können Sie stattdessen das AWS -Toolkit for Eclipse installieren.

Für die Verbraucheranwendung ist die Kinesis Client Library (KCL) Version 1.2.1 oder höher erforderlich, die Sie GitHub unter Kinesis Client Library (Java) beziehen können.

Nächste Schritte

Erstellen Sie einen Datenstream

Erstellen Sie einen Datenstream

Im ersten Schritt von Tutorial: Verarbeiten Sie Aktiendaten in Echtzeit mit KPL und KCL 1.x erstellen Sie den Stream, der in den weiteren Schritten genutzt wird.

So erstellen Sie einen Stream

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Kinesis-Konsole unter https://console.aws.amazon.com/kinesis.
2. Klicken Sie im Navigationsbereich auf Data Streams (Daten-Streams).
3. Erweitern Sie in der Navigationsleiste die Regionsauswahl und wählen Sie eine Region aus.
4. Wählen Sie Create Kinesis Stream (Kinesis-Stream erstellen).
5. Geben Sie einen Namen für Ihren Stream ein (z. B. **`StockTradeStream`**).
6. Geben Sie **1** die Anzahl der Shards ein, lassen Sie aber Estimate the number of shards you'll need (Schätzung der Anzahl der Shards, die Sie benötigen) reduziert.
7. Wählen Sie Create Kinesis Stream (Kinesis-Stream erstellen).

Auf der Seite mit der Auflistung der Kinesis Streams ist der Status Ihres Streams während des Erstellens auf `CREATING` gesetzt. Sobald der Stream verwendet werden kann, ändert sich der Status in `ACTIVE`. Wählen Sie den Namen des Streams aus. Auf der angezeigten Seite zeigt die Registerkarte Details eine Zusammenfassung Ihrer Streams-Konfiguration an. Der Abschnitt Monitoring (Überwachung) zeigt Überwachungsinformationen für den Stream an.

Zusätzliche Informationen zu Shards

Wenn Sie Kinesis Data Streams außerhalb dieses Tutorials verwenden, sollten Sie das Erstellen des Streams sorgfältiger planen. Bei der Bereitstellung von Shards sollten Sie vom höchsten erwarteten Bedarf ausgehen. In unserem Beispielszenario kommt es an der US-Börse tagsüber zu Datenverkehrsspitzen (Eastern Time). Der Bedarf sollte so geschätzt werden, dass diesen Verkehrsspitzen Rechnung getragen wird. Sie können dann entweder für den höchst möglichen Bedarf vorsorgen oder Ihren Stream an die Schwankungen anpassen.

Ein Shard ist eine Einheit für die Durchsatzkapazität. Erweitern Sie auf der Seite Kinesis Stream erstellen Schätzen Sie die Anzahl der Shards, die Sie brauchen. Geben Sie entsprechend der folgenden Informationen die durchschnittliche Datensatzgröße, die maximale Anzahl der Datensätze, die pro Sekunde geschrieben werden, und die Anzahl der Konsumentenanwendungen an:

Durchschnittliche Datensatzgröße

Eine Schätzung der berechneten durchschnittlichen Größe Ihrer Datensätze. Wenn Sie diesen Wert nicht kennen, verwenden Sie die geschätzte maximale Datensatzgröße.

Max. geschriebene Datensätze

Berücksichtigen Sie die Anzahl der Entitäten, die Daten bereitstellen, und die ungefähre Anzahl der jeweils erstellten Datensätze pro Sekunde. Beispiel: Wenn Sie Börsendaten von 20 Handelsservern erhalten und jeder pro Sekunde 250 Handelsabschlüsse generiert, beträgt die Anzahl der Handelsabschlüsse (Datensätze) 5.000/Sekunde.

Anzahl der Konsumentenanwendungen

Die Anzahl der Anwendungen, die unabhängig voneinander Daten aus dem Stream auslesen, die Stream-Daten unterschiedlich verarbeiten und unterschiedliche Ausgaben generieren. Es können mehrere Instances einer Anwendung auf verschiedenen Rechnern (d. h. in einem Cluster) ausgeführt werden, sodass ein großer Daten-Stream verarbeitet werden kann.

Wenn der gezeigte Schätzwert der Shards das aktuelle Shard-Limit übersteigt, müssen Sie ggf. eine Erhöhung des Limits beantragen, bevor Sie einen Stream mit dieser Anzahl an Shards erstellen können. Nutzen Sie zum Beantragen einer Erhöhung des Shard-Limits das Formular für Limits der Kinesis Data Streams. Weitere Informationen zu Streams und Shards finden Sie unter Kinesis-Datenstreams erstellen und verwalten.

Nächste Schritte

Erstellen Sie eine IAM-Richtlinie und einen Benutzer

Erstellen Sie eine IAM-Richtlinie und einen Benutzer

Bewährte Sicherheitsmethoden für AWS schreiben die Verwendung detaillierter Berechtigungen vor, um den Zugriff auf verschiedene Ressourcen zu kontrollieren. AWS Identity and Access Management (IAM) ermöglicht Ihnen die Verwaltung von Benutzern und Benutzerberechtigungen in. AWS Eine IAM-Richtlinie führt explizit zulässige Aktionen sowie Ressourcen auf, für die diese Aktionen relevant sind.

Im Folgenden werden die Berechtigungen für einen Produzenten und Konsumenten von Kinesis Data Streams aufgelistet, die mindestens erforderlich sind.

Produzent

Aktionen	Ressource	Zweck
`DescribeStream` , `DescribeStreamSummary` , `DescribeStreamConsumer`	Kinesis Data Streams	Bevor er versucht, Datensätze zu schreiben, prüft der Produ vorhanden und aktiv ist, ob die Shards im Stream enthalten einen Consumer hat.
`SubscribeToShard` , `RegisterStreamConsumer`	Kinesis Data Streams	Abonnieren und Registrieren eines Konsumenten bei einem Shard.
`PutRecord` , `PutRecords`	Kinesis Data Streams	Schreiben von Datensätzen in Kinesis Data Streams.

Konsument

Aktionen	Ressource	Zweck
`DescribeStream`	Kinesis Data Streams	Vor dem Versuch, Daten zu lesen, prüft der Konsument, ob und aktiv ist und ob Shards im Stream enthalten sind.
`GetRecords` , `GetShardIterator`	Kinesis Data Streams	Lesen von Datensätzen aus einem Shard von Kinesis Data
`CreateTable` , `DescribeTable` , `GetItem`, `PutItem`, `Scan`, `UpdateItem`	Amazon-DynamoDB-Tabelle.	Wenn der Konsument mit der Kinesis Client Library (KCL) e er Berechtigungen für die DynamoDB-Tabelle, um den Vera Anwendung zu verfolgen. Der erste gestartete Konsument e
`DeleteItem`	Amazon-DynamoDB-Tabelle.	Für den Fall, dass der Verbraucher split/merge Operationen Streams Streams-Shards ausführt.
`PutMetricData`	CloudWatch Amazon-Protokoll	Die KCL lädt auch Metriken hoch CloudWatch, die für die Ü Anwendung nützlich sind.

Für diese Anwendung erstellen Sie eine einzelne IAM-Richtlinie, die alle vorstehenden Berechtigungen gewährt. In der Praxis empfiehlt es sich möglicherweise, zwei Richtlinien zu erstellen: eine für Produzenten und eine für Konsumenten.

So erstellen Sie eine IAM-Richtlinie

1. Suchen Sie den Amazon-Ressourcennamen (ARN) für den neuen Stream. Sie finden diesen ARN als Stream ARN (Stream-ARN) oben auf der Registerkarte Details. Das ARN-Format sieht folgendermaßen aus:

   ```
   arn:aws:kinesis:region:account:stream/name
   ```

   Region

   Der Regionscode, beispielsweise `us-west-2`. Weitere Informationen finden Sie unter Regionen und Verfügbarkeitskonzepte.

   Konto

   Die AWS Konto-ID, wie in den Kontoeinstellungen angezeigt.

   Name

   Der Name des Streams von Erstellen Sie einen Datenstream, hier `StockTradeStream`.

2. Bestimmen Sie den ARN für die DynamoDB-Tabelle, die vom Konsumenten verwendet wird (und von der ersten Konsumenten-Instance erstellt wird). Er muss das folgende Format aufweisen:

   ```
   arn:aws:dynamodb:region:account:table/name
   ```

   Region und Konto stammen vom selben Ort wie im vorherigen Schritt. Hier ist der Name jedoch der Name der Tabelle, die von der Verbraucheranwendung erstellt und verwendet wird. Die vom Konsumenten verwendete KCL nutzt den Anwendungsnamen als Tabellennamen. Verwenden Sie `StockTradesProcessor`, dies ist der Anwendungsname, der später genutzt wird.

3. Wählen Sie in der IAM-Konsole unter Policies (https://console.aws.amazon.com/iam/home#policies) die Option Create policy aus. Wenn Sie zum ersten Mal mit IAM-Richtlinien arbeiten, wählen Sie Erste Schritte, Richtlinie erstellen aus.
4. Wählen Sie Select (Auswählen) neben Policy Generator (Richtliniengenerator) aus.
5. Wählen Sie Amazon Kinesis als AWS Service.

6. Legen Sie `DescribeStream`, `GetShardIterator`, `GetRecords`, `PutRecord` und `PutRecords` als zulässige Aktionen fest.
7. Geben Sie den in Schritt 1 erstellten ARN ein.
8. Verwenden Sie Add Statement (Statement hinzufügen) für die folgenden Elemente:

AWS Dienst	Aktionen	ARN
Amazon DynamoDB	`CreateTable` , `DeleteItem` , `DescribeTable` , `GetItem`, `PutItem`, `Scan`, `UpdateItem`	Der ARN, den Sie in Schritt 2 erstellt haben.
Amazon CloudWatch	`PutMetricData`	*

   Das Sternchen (*), das bei der Angabe einer ARN verwendet wird, ist nicht erforderlich. In diesem Fall liegt das daran, dass es keine bestimmte Ressource gibt, für die die `PutMetricData` Aktion aufgerufen wird. CloudWatch
9. Wählen Sie Next Step (Weiter) aus.
10. Ändern Sie Policy Name (Richtlinienname) in `StockTradeStreamPolicy`, prüfen Sie den Code und wählen sie Create Policy (Richtlinie erstellen) aus.

Das erstellte Richtliniendokument sollte etwa wie folgt aussehen:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt123",
            "Effect": "Allow",
            "Action": [
                "kinesis:DescribeStream",
                "kinesis:PutRecord",
                "kinesis:PutRecords",
```

```
                "kinesis:GetShardIterator",
                "kinesis:GetRecords",
                "kinesis:ListShards",
                "kinesis:DescribeStreamSummary",
                "kinesis:RegisterStreamConsumer"
            ],
            "Resource": [
                "arn:aws:kinesis:us-west-2:111122223333:stream/StockTradeStream"
            ]
        },
        {
            "Sid": "Stmt234",
            "Effect": "Allow",
            "Action": [
                "kinesis:SubscribeToShard",
                "kinesis:DescribeStreamConsumer"
            ],
            "Resource": [
                "arn:aws:kinesis:us-west-2:111122223333:stream/StockTradeStream/
*"
            ]
        },
        {
            "Sid": "Stmt456",
            "Effect": "Allow",
            "Action": [
                "dynamodb:*"
            ],
            "Resource": [
                "arn:aws:dynamodb:us-west-2:111122223333:table/
StockTradesProcessor"
            ]
        },
        {
            "Sid": "Stmt789",
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricData"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
```

```
}
```

So erstellen Sie einen IAM-Benutzer

1. Öffnen Sie unter https://console.aws.amazon.com/iam/ die IAM-Konsole.
2. Wählen Sie auf der Seite Users (Benutzer) die Option Add user (Benutzer hinzufügen) aus.
3. Geben Sie `StockTradeStreamUser` als Benutzername ein.
4. Wählen Sie für Access type (Zugriffstyp) die Option Programmatic access (Programmgesteuerter Zugriff) und wählen Sie dann Next: Permissions (Weiter: Berechtigungen).
5. Wählen Sie Vorhandene Richtlinien direkt zuordnen.
6. Suche die von Ihnen erstellte Richtlinie dem Namen nach. Markieren Sie das Kontrollkästchen links neben dem Richtliniennamen und wählen Sie dann Next: Review (Weiter: Prüfen).
7. Überprüfen Sie die Details und die Zusammenfassung und wählen Sie dann Create user (Benutzer erstellen) aus.
8. Kopieren Sie die Access key ID (Zugriffsschlüssel-ID) und speichern Sie sie privat. Wählen Sie unter Secret access key (Geheimer Zugriffsschlüssel) die Option Show (Anzeigen) und speichern Sie auch diesen Schlüssel privat.
9. Fügen Sie die Zugriffs- und Geheimschlüssel in eine lokalen Datei an einem sichern Ort ein, auf den nur Sie Zugriff haben. Erstellen Sie für diese Anwendung eine Datei namens `~/.aws/credentials` (mit strikten Berechtigungen). Die Datei sollte das folgende Format aufweisen:

   ```
   [default]
   aws_access_key_id=access key
   aws_secret_access_key=secret access key
   ```

So fügen Sie einem Benutzer eine IAM-Richtlinie hinzu

1. Klicken Sie in der IAM;-Konsole auf Richtlinien und wählen Sie Richtlinienaktionen aus.
2. Klicken Sie auf `StockTradeStreamPolicy` und Attach (Verknüpfen).
3. Wählen Sie `StockTradeStreamUser` und Attach Policy (Richtlinie anfügen) aus.

Nächste Schritte

Laden Sie den Implementierungscode herunter und erstellen Sie ihn

Laden Sie den Implementierungscode herunter und erstellen Sie ihn

Für the section called “Tutorial: Verarbeiten Sie Aktiendaten in Echtzeit mit KPL und KCL 1.x” wird Skeleton-Code bereitgestellt. Er enthält eine Stub-Implementierung für die Übernahme des Wertpapierdaten-Streams (Produzent) und die Verarbeitung der Daten (Verbraucher). Das folgende Verfahren zeigt, wie die Implementierung abgeschlossen wird.

So laden und erstellen Sie den Implementierungscode

1. Laden Sie den Quellcode auf den Computer herunter.
2. Erstellen Sie mit dem Quellcode unter Einhaltung der bereitgestellten Verzeichnisstruktur ein Projekt in der bevorzugten IDE.
3. Fügen Sie dem Projekt die folgenden Bibliotheken hinzu:
   - Amazon Kinesis Client Library (KCL)
   - AWS SDK
   - Apache HttpCore
   - Apache HttpClient
   - Apache Commons Lang
   - Apache Commons Logging
   - Guava (Google-Kernbibliotheken für Java)
   - Jackson Annotations
   - Jackson Core
   - Jackson Databind
   - Jackson Dataformat: CBOR
   - Joda Time
4. Je nach IDE wird das Projekt möglicherweise automatisch erstellt. Wenn nicht, erstellen Sie es mit den für Ihre IDE erforderlichen Schritten.

Wenn Sie diese Schritte erfolgreich abgeschlossen haben, können Sie zum nächsten Abschnitt wechseln, the section called “Implementieren Sie den Produzenten”. Wenn der Build Fehler erzeugt, müssen Sie diese untersuchen und beheben, bevor Sie fortfahren.

Nächste Schritte

Implementieren Sie den Produzenten

Die Anwendung im Tutorial: Verarbeiten Sie Aktiendaten in Echtzeit mit KPL und KCL 1.x verwendet das reale Szenario einer Überwachung des Wertpapierhandels. Im Folgenden wird kurz erläutert, wie dieses Szenario zum Produzenten und der unterstützenden Codestruktur zugeordnet wird.

Überprüfen Sie die folgenden Informationen in Bezug auf den Quellcode.

StockTrade Klasse

Ein bestimmter Wertpapierhandel wird durch eine Instance der `StockTrade`-Klasse dargestellt. Diese Instance enthält folgende Attribute: Tickersymbol, Preis, Anzahl der Anteile, Art des Handels (Kauf oder Verkauf) und ID zur eindeutigen Identifizierung der Handelsaktion. Dieser Klasse wird für Sie implementiert.

Stream-Datensatz

Ein Stream ist eine Sequenz von Datensätzen. Ein Datensatz ist die Serialisierung einer `StockTrade`-Instance im JSON-Format. Beispiel:

```
{
  "tickerSymbol": "AMZN",
  "tradeType": "BUY",
  "price": 395.87,
  "quantity": 16,
  "id": 3567129045
}
```

StockTradeGenerator Klasse

`StockTradeGenerator` verfügt über eine Methode namens `getRandomTrade()`, die bei Aufruf Daten eines zufällig generierten Wertpapierhandels zurückgibt. Dieser Klasse wird für Sie implementiert.

StockTradesWriter Klasse

Die `main`-Methode des Produzenten, `StockTradesWriter`, ruft kontinuierlich eine zufällige Handelsaktion ab und sendet die Daten an Kinesis Data Streams, indem sie die folgenden Aufgaben durchführt:

1. Lesen des Stream- und Regionsnamen als Eingabe.

2. Erstellen eines `AmazonKinesisClientBuilder`.
3. Verwenden des Client Builder, um die Region, die Anmeldeinformationen und die Client-Konfiguration festzulegen.
4. Erstellen eines `AmazonKinesis`-Clients mit dem Client-Builder.
5. Sicherstellen, dass der Stream vorhanden und aktiv ist (wenn nicht, kommt es zu einer Beendigung mit Fehler).
6. Aufrufen der `StockTradeGenerator.getRandomTrade()`-Methode in einer Dauerschleife und anschließend Aufruf der `sendStockTrade`-Methode, um die Handelsdaten alle 100 Millisekunden an den Stream zu senden.

Die `sendStockTrade`-Methode der `StockTradesWriter`-Klasse hat den folgenden Code:

```
private static void sendStockTrade(StockTrade trade, AmazonKinesis kinesisClient,
 String streamName) {
    byte[] bytes = trade.toJsonAsBytes();
    // The bytes could be null if there is an issue with the JSON serialization by
 the Jackson JSON library.
    if (bytes == null) {
        LOG.warn("Could not get JSON bytes for stock trade");
        return;
    }

    LOG.info("Putting trade: " + trade.toString());
    PutRecordRequest putRecord = new PutRecordRequest();
    putRecord.setStreamName(streamName);
    // We use the ticker symbol as the partition key, explained in the Supplemental
 Information section below.
    putRecord.setPartitionKey(trade.getTickerSymbol());
    putRecord.setData(ByteBuffer.wrap(bytes));

    try {
        kinesisClient.putRecord(putRecord);
    } catch (AmazonClientException ex) {
        LOG.warn("Error sending record to Amazon Kinesis.", ex);
    }
}
```

Beachten Sie die folgende Code-Struktur:

- Die `PutRecord` API erwartet ein Byte-Array, und Sie müssen es in das JSON-Format `trade` konvertieren. Diese einzelne Codezeile führt die Operation aus:

```
byte[] bytes = trade.toJsonAsBytes();
```

- Bevor Sie die Handelsdaten senden können, erstellen Sie eine neue `PutRecordRequest`-Instance (namens `putRecord` in diesem Fall):

  ```
  PutRecordRequest putRecord = new PutRecordRequest();
  ```

  Jeder `PutRecord`-Aufruf erfordert den Namen des Streams, einen Partitionsschlüssel und einen Daten-Blob. Der folgende Code füllt diese Felder im `putRecord`-Objekt mit dessen `setXxxx()`-Methoden:

  ```
  putRecord.setStreamName(streamName);
  putRecord.setPartitionKey(trade.getTickerSymbol());
  putRecord.setData(ByteBuffer.wrap(bytes));
  ```

  Im Beispiel wird ein Stock Ticket als Partitionsschlüssel verwendet, wodurch der Datensatz einem bestimmten Shard zugeordnet wird. In der Praxis sollten Sie Hunderte oder gar Tausende von Partitionsschlüsseln pro Shard haben, sodass die Datensätze in Ihrem Stream gleichmäßig verteilt sind. Weitere Informationen zum Hinzufügen von Daten zu einem Stream finden Sie unter Daten zu einem Stream hinzufügen.

  `putRecord` ist nun für das Senden an den Client bereit (`put`-Operation):

  ```
  kinesisClient.putRecord(putRecord);
  ```

- Eine Fehlerüberprüfung und Protokollierung sind immer nützliche Ergänzungen. Dieser Code protokolliert Fehlerbedingungen:

  ```
  if (bytes == null) {
      LOG.warn("Could not get JSON bytes for stock trade");
      return;
  }
  ```

  Fügen Sie den try/catch Block rund um die `put` Operation hinzu:

  ```
  try {
         kinesisClient.putRecord(putRecord);
  } catch (AmazonClientException ex) {
  ```

```
        LOG.warn("Error sending record to Amazon Kinesis.", ex);
}
```

Der Grund besteht darin, dass eine Kinesis-Data-Streams-`put`-Operation aufgrund eines Netzwerkfehlers fehlschlagen kann oder gedrosselt wird, weil die Durchsatzgrenze des Streams erreicht wird. Wir empfehlen, Ihre Wiederholungsrichtlinie für `put` Operationen sorgfältig zu überdenken, um Datenverlust zu vermeiden, z. B. die Verwendung eines Wiederholungsversuchs.

- Eine Statusprotokollierung ist hilfreich, wenn auch optional:

```
LOG.info("Putting trade: " + trade.toString());
```

Der hier gezeigte Produzent verwendet die API-Funktionalität von Kinesis Data Streams für einzelne Datensätze `PutRecord`. In der Praxis ist es oft effizienter, die Eignung von `PutRecords` für mehrere Datensätze zu nutzen und mehrere Datensatzstapel gleichzeitig zu senden, wenn ein Produzent viele Datensätze erstellt. Weitere Informationen finden Sie unter Daten zu einem Stream hinzufügen.

So führen Sie den Produzenten aus

1. Stellen Sie sicher, dass der Zugriffsschlüssel und das geheime Schlüsselpaar, die vorher (beim Erstellen des IAM-Benutzers) abgerufen wurden, in der Datei `~/.aws/credentials` gespeichert sind.
2. Führen Sie die `StockTradeWriter`-Klasse mit den folgenden Argumenten aus:

```
StockTradeStream us-west-2
```

Wenn Sie Ihren Stream in einer anderen Region als `us-west-2` erstellt haben, müssen Sie stattdiesen hier diese Region angeben.

Die Ausgabe sollte folgendermaßen oder ähnlich aussehen:

```
Feb 16, 2015 3:53:00 PM
com.amazonaws.services.kinesis.samples.stocktrades.writer.StockTradesWriter
 sendStockTrade
INFO: Putting trade: ID 8: SELL 996 shares of BUD for $124.18
Feb 16, 2015 3:53:00 PM
```

```
com.amazonaws.services.kinesis.samples.stocktrades.writer.StockTradesWriter
 sendStockTrade
INFO: Putting trade: ID 9: BUY 159 shares of GE for $20.85
Feb 16, 2015 3:53:01 PM
com.amazonaws.services.kinesis.samples.stocktrades.writer.StockTradesWriter
 sendStockTrade
INFO: Putting trade: ID 10: BUY 322 shares of WMT for $90.08
```

Ihr Stream für die Wertpapierdaten wird nun von Kinesis Data Streams eingespeist.

Nächste Schritte

Implementieren Sie den Verbraucher

Implementieren Sie den Verbraucher

Die Konsumentenanwendung in Tutorial: Verarbeiten Sie Aktiendaten in Echtzeit mit KPL und KCL 1.x verarbeitet kontinuierlich den in erstellten Stream mit Wertpapiertransaktionen. Sie gibt dann für jede Minute die beliebtesten Aktien aus, die gekauft und verkauft wurden. Die Anwendung setzt auf Kinesis Client Library (KCL) auf, die viele der mühsamen Arbeiten einer Verbraucheranwendung übernimmt. Weitere Informationen finden Sie unter Entwickeln Sie KCL 1.x-Verbraucher.

Überprüfen Sie die folgenden Informationen in Bezug auf den Quellcode.

StockTradesProcessor Klasse

Hauptklasse des Konsumenten, die für Sie bereitgestellt wird und folgende Aufgaben übernimmt:

- Liest die Namen von Anwendung, Stream und Region, die als Argumente übergeben werden
- Liest Anmeldeinformationen aus `~/.aws/credentials`
- Erstellt eine `RecordProcessorFactory`-Instance für die Instances von `RecordProcessor`, implementiert von einer `StockTradeRecordProcessor`-Instance.
- Erstellt einen KCL-Auftragnehmer mit der `RecordProcessorFactory`-Instance und eine Standardkonfiguration samt Stream-Name, Anmeldeinformationen und Anwendungsname.
- Die Worker erstellt für jeden Shard (der dieser Konsumenten-Instance zugeordnet ist) einen neuen Thread, der die Datensätze in einer Schleife aus Kinesis Data Streams liest. Anschließend wird die `RecordProcessor`-Instance aufgerufen, um die empfangenen Datensatzstapel zu verarbeiten.

StockTradeRecordProcessor Klasse

Implementierung der `RecordProcessor`-Instance, die wiederum drei erforderliche Methoden implementiert: `initialize`, `processRecords` und `shutdown`.

Wie an den Namen zu erkennen ist, werden `initialize` und `shutdown` von der Kinesis Client Library verwendet, um dem Datensatzprozessor mitzuteilen, wann er mit dem Empfang von Datensätzen beginnen bzw. wann er diesen stoppen soll, sodass er entsprechende anwendungsspezifische Einrichtungs- und Beendigungsaufgaben ausführen kann. Der Code hierfür wird für Sie bereitgestellt. Die wesentliche Verarbeitung erfolgt mit der `processRecords` Methode, die wiederum `processRecord` für die einzelnen Datensätze nutzt. Die letztgenannte Methode wird als nahezu leeres Code-Skelett bereitgestellt, das im nächsten Schritt näher erläutert und von Ihnen implementiert wird.

Darüber hinaus hervorzuheben ist die Implementierung von Support-Methoden für `processRecord`: `reportStats` und `resetStats`, die im ursprünglichen Quellcode leer sind.

Die `processRecords`-Methode wurde für Sie implementiert und führt die folgenden Schritte aus:

- Ruft `processRecord` für jeden übergebenen Datensatz auf.
- Ruft `reportStats()` zum Drucken der neuesten Statistiken auf, wenn seit dem letzten Bericht mindestens 1 Minute vergangen ist, und dann `resetStats()`, um die Statistiken zu löschen, damit das nächste Intervall nur neue Datensätze enthält.
- Legt den Zeitpunkt für die nächste Berichterstellung fest.
- Ruft `checkpoint()` auf, wenn seit dem letzten Prüfpunkt mindestens 1 Minute vergangen ist.
- Legt den Zeitpunkt für das nächste Checkpointing fest.

Diese Methode verwendet für das Checkpointing und die Berichterstellung ein Intervall von 60 Sekunden. Weitere Informationen zum Checkpointing finden Sie unter Zusätzliche Informationen über den Verbraucher.

StockStats Klasse

Diese Klasse stellt eine Datenaufbewahrung und eine Nachverfolgung von Statistiken für die beliebtesten Aktien bereit. Dieser Code wird für Sie bereitgestellt und enthält folgende Methoden:

- `addStockTrade(StockTrade)`: Fügt den angegebenen `StockTrade` in die ausgeführten Statistiken ein.
- `toString()`: Gibt die Statistiken in einer formatierten Zeichenfolge zurück.

Diese Klasse verfolgt die beliebtesten Aktien, indem sie fortlaufend die Gesamtzahl der Trades für jede Aktie und deren maximale Anzahl zählt. Sie aktualisiert diese Werte, sobald eine neue Handelstransaktion empfangen wird.

Fügen Sie Code zu den Methoden der `StockTradeRecordProcessor`-Klasse hinzu, wie in den folgenden Schritten gezeigt.

So implementieren Sie den Konsumenten

1. Implementieren Sie die `processRecord`-Methode, indem Sie ein richtig bemessenes `StockTrade`-Objekt instanziieren und die Datensatzdaten zu diesem hinzufügen, sodass im Falle eines Problems eine Warnung protokolliert wird.

```
StockTrade trade = StockTrade.fromJsonAsBytes(record.getData().array());
if (trade == null) {
    LOG.warn("Skipping record. Unable to parse record into StockTrade. Partition
 Key: " + record.getPartitionKey());
    return;
}
stockStats.addStockTrade(trade);
```

2. Implementieren Sie eine einfache `reportStats`-Methode. Sie können das Ausgabeformat an Ihre Bedürfnisse anpassen.

```
System.out.println("****** Shard " + kinesisShardId + " stats for last 1 minute
 ******\n" +
                   stockStats + "\n" +

 "****************************************************************\n");
```

3. Implementieren Sie abschließend die `resetStats`-Methode, die eine neue `stockStats`-Instance erstellt.

```
stockStats = new StockStats();
```

So führen Sie den Konsumenten aus

1. Führen Sie den unter erstellten Produzenten aus, um simulierte Wertpapiertransaktionsdatensätze in den Stream zu schreiben.

2. Stellen Sie sicher, dass der Zugriffsschlüssel und das geheime Schlüsselpaar, die vorher (beim Erstellen des IAM-Benutzers) abgerufen wurden, in der Datei `~/.aws/credentials` gespeichert sind.
3. Führen Sie die `StockTradesProcessor`-Klasse mit den folgenden Argumenten aus:

   ```
   StockTradesProcessor StockTradeStream us-west-2
   ```

   Beachten Sie, dass Sie, wenn Sie Ihren Stream in einer anderen Region als `us-west-2` erstellt haben, stattdiesen diese Region hier angeben müssen.

Nach einer Minute sollen Sie eine Ausgabe ähnlich der folgenden sehen, die anschließend einmal pro Minute aktualisiert wird:

```
****** Shard shardId-000000000001 stats for last 1 minute ******
Most popular stock being bought: WMT, 27 buys.
Most popular stock being sold: PTR, 14 sells.
************************************************************
```

Zusätzliche Informationen über den Verbraucher

Wenn Sie mit den Vorteilen der Kinesis Client Library vertraut sind, die unter Entwickeln Sie KCL 1.x-Verbraucher und anderswo erörtert werden, fragen Sie sich möglicherweise, warum Sie sie hier nutzen sollten. Obwohl Sie für die Verarbeitung nur einen einzelnen Shard-Stream und eine einzelne Konsumenten-Instance benötigen, ist es immer noch einfacher, den Konsumenten unter Verwendung der KCL zu implementieren. Vergleichen Sie die Implementierungsschritte im Produzentenabschnitt mit denen für den Konsumenten und Sie werden feststellen, dass die Implementierung eines Konsumenten vergleichsweise einfach ist. Dies liegt hauptsächlich an den von der KCL bereitgestellten Services.

In dieser Anwendung konzentrieren Sie sich auf die Implementierung einer Datensatzprozessor-Klasse, die einzelne Datensätze verarbeiten kann. Sie müssen sich keine Gedanken darüber machen, wie die Datensätze aus Kinesis Data Streams abgerufen werden. Die KCL ruft die Datensätze ab und den Datensatzprozessor auf, wenn neue Datensätze verfügbar sind. Sie müssen sich zudem keine Gedanken über die Anzahl der vorhandenen Shards und Konsumenten-Instances machen. Wenn der Stream skaliert wird, müssen Sie Ihre Anwendung nicht neu schreiben, damit mehr als ein Shard oder eine Konsumenten-Instance verwaltet werden können.

Der Begriff Checkpointing bedeutet, den Punkt im Stream bis hin zu den bisher verbrauchten und verarbeiteten Datensätzen aufzuzeichnen. Wenn die Anwendung abstürzt, wird der Stream von diesem Punkt aus gelesen und nicht vom Anfang des Streams. Das Checkpointing sowie zugehörige Entwurfsmuster und bewährte Methoden sind nicht Gegenstand dieses Kapitels. Es ist jedoch etwas, das Ihnen in Produktionsumgebungen begegnen könnte.

Wie Sie in erfahren haben, verwenden die `put`-Operationen in der API für Kinesis Data Streams einen Partitionsschlüssel als Eingabe. Kinesis Data Streams verwendet einen Partitionsschlüssel zum Aufteilen von Datensätzen auf mehrere Shards (wenn mehr als ein Shard im Stream vorhanden ist). Derselbe Partitionsschlüssel leitet immer an denselben Shard weiter. Dadurch kann der Konsument, der einen bestimmten Shard verarbeitet, davon ausgehen, dass Datensätze mit demselben Partitionsschlüssel nur an ihn gesendet werden und Datensätze mit diesem Partitionsschlüssel nicht bei einem anderen Konsumenten landen. Deshalb kann der Worker eines Konsumenten alle Datensätze mit demselben Partitionsschlüssel aggregieren, ohne zu befürchten, dass erforderliche Daten fehlen.

In dieser Anwendung findet keine intensive Verarbeitung der Datensätze durch den Konsumenten statt. Deshalb können Sie einen Shard verwenden und die Verarbeitung im selben Thread wie der KCL-Thread durchführen. In der Praxis sollten Sie allerdings zunächst die Anzahl der Shards skalieren. Gelegentlich kann es vorkommen, dass Sie die Verarbeitung an einen anderen Thread übergeben oder einen Thread-Pool nutzen möchten, wenn eine intensive Datensatzverarbeitung ansteht. Dadurch kann die KCL die neuen Datensätze schneller abrufen, während die anderen Threads parallel dazu die Datensätze verarbeiten. Multithread-Design ist nicht trivial und sollte mit fortgeschrittenen Techniken angegangen werden. Daher ist die Erhöhung der Anzahl Ihrer Shards in der Regel die effektivste Methode zur Skalierung.

Nächste Schritte

(Optional) Erweitern Sie die Zahl der Verbraucher

(Optional) Erweitern Sie die Zahl der Verbraucher

Möglicherweise ist die Anwendung Tutorial: Verarbeiten Sie Aktiendaten in Echtzeit mit KPL und KCL 1.x bereits ausreichend für Ihre Zwecke. Dieser optionale Abschnitt zeigt, wie Sie den Konsumentencode erweitern können, um einem komplexeren Szenario gerecht zu werden.

Wenn Sie minütlich über die größten Verkaufsaufträge informiert werden möchten, können Sie die `StockStats`-Klasse an drei Stellen bearbeiten.

So erweitern Sie den Konsumenten

1. Fügen Sie neue Instance-Variablen hinzu:

```
// Ticker symbol of the stock that had the largest quantity of shares sold
private String largestSellOrderStock;
// Quantity of shares for the largest sell order trade
private long largestSellOrderQuantity;
```

2. Fügen Sie folgenden Code zu hinz `addStockTrade`:

```
if (type == TradeType.SELL) {
    if (largestSellOrderStock == null || trade.getQuantity() >
largestSellOrderQuantity) {
        largestSellOrderStock = trade.getTickerSymbol();
        largestSellOrderQuantity = trade.getQuantity();
    }
}
```

3. Ändern Sie die `toString`-Methode, um die zusätzlichen Informationen zu drucken:

```
public String toString() {
    return String.format(
            "Most popular stock being bought: %s, %d buys.%n" +
            "Most popular stock being sold: %s, %d sells.%n" +
            "Largest sell order: %d shares of %s.",
            getMostPopularStock(TradeType.BUY),
getMostPopularStockCount(TradeType.BUY),
            getMostPopularStock(TradeType.SELL),
getMostPopularStockCount(TradeType.SELL),
            largestSellOrderQuantity, largestSellOrderStock);
}
```

Wenn Sie den Konsumenten jetzt ausführen (führen Sie auch den Produzenten), sollten Sie eine Ausgabe ähnlich der folgenden sehen:

```
****** Shard shardId-000000000001 stats for last 1 minute ******
 Most popular stock being bought: WMT, 27 buys.
 Most popular stock being sold: PTR, 14 sells.
 Largest sell order: 996 shares of BUD.
 ************************************************************
```

Nächste Schritte

Bereinigen von Ressourcen

Bereinigen von Ressourcen

Da Sie für die Nutzung des Kinesis Data Streams zahlen, sollten Sie diesen ebenso wie die entsprechende Amazon-DynamoDB-Tabelle löschen, wenn sie nicht mehr benötigt werden. Für einen aktiven Stream fallen auch dann nominale Gebühren an, wenn Sie keine Datensätze senden oder abrufen. Der Grund hierfür ist, dass ein aktiver Stream durch kontinuierlichen „Horchen“ darauf, ob neue Datensätze oder Anforderungen zum Abrufen von Datensätzen eingehen, Ressourcen verbraucht.

So löschen Sie den Stream und die Tabelle

1. Fahren Sie alle laufenden Produzenten und Konsumenten herunter.
2. Öffnen Sie die Kinesis-Konsole unter https://console.aws.amazon.com/kinesis.
3. Wählen Sie den Stream aus, den Sie für diese Anwendung erstellt haben (`StockTradeStream`).
4. Wählen Sie Delete Stream (Stream löschen) aus.
5. Öffnen Sie die DynamoDB-Konsole unter. https://console.aws.amazon.com/dynamodb/
6. Löschen Sie die `StockTradesProcessor`-Tabelle.

Zusammenfassung

Die Verarbeitung einer großen Datenmenge nahezu in Echtzeit erfordert weder das Schreiben von kompliziertem Code noch die Entwicklung einer riesigen Infrastruktur. Es ist genauso einfach wie die Schreiblogik, eine kleine Datenmenge zu verarbeiten (wie Schreiben`processRecord(Record)`), aber Kinesis Data Streams zu verwenden, um so zu skalieren, dass es für eine große Menge an Streaming-Daten funktioniert. Sie müssen sich keine Gedanken über die Skalierung der Verarbeitung machen, da Kinesis Data Streams das für Sie übernimmt. Sie müssen lediglich Ihre Streaming-Datensätze an Kinesis Data Streams senden und eine Logik für die Verarbeitung neu empfangener Datensätze schreiben.

Nachfolgend einige mögliche Erweiterungen für diese Anwendung.

Shard-übergreifende Aggregation

Derzeit erhalten Sie Statistiken, die aus der Aggregation von Datensätzen resultieren, die von einem einzelnen Auftragnehmer eines einzelnen Shards empfangen werden. (Ein Shard kann jeweils nur von einem Auftragnehmer in einer einzelnen Anwendung verarbeitet werden.) Möglicherweise möchten Sie, wenn Sie eine Skalierung durchführen und mehr als einen Shard haben, eine Shard-übergreifende Aggregation vornehmen. Dies kann mit einer Pipeline-Architektur realisiert werden, bei der die Ausgabe eines jeden Auftragnehmers in einen anderen Stream mit einem einzelnen Shard eingespeist wird, der von einem Auftragnehmer verarbeitet wird, der die Ausgaben der ersten Phase aggregiert. Da die Daten aus der ersten Phase begrenzt sind (eine Ausgabe pro Minute pro Shard), können sie problemlos von nur einem Shard verarbeitet werden.

Skalierung

Wenn der Stream skaliert wird, damit mehr Shards zur Verfügung stehen (da viele Produzenten Daten senden), geschieht dies dadurch, dass mehr Worker hinzugefügt werden. Sie können die Auftragnehmer in EC2-Instances ausführen und Auto-Scaling-Gruppen nutzen.

Verwenden Sie Konnektoren zu Amazon S3 S3/ DynamoDB/Amazon Redshift/Storm

Da ein Stream kontinuierlich verarbeitet wird, kann seine Ausgabe an andere Ziele gesendet werden. AWS bietet Konnektoren zur Integration von Kinesis Data Streams mit anderen AWS Diensten und Tools von Drittanbietern.

Nächste Schritte

- Weitere Informationen zur Verwendung der API-Operationen von Kinesis Data Streams finden Sie unter Entwickeln Sie Produzenten mithilfe der Amazon Kinesis Data Streams Streams-API mit dem AWS SDK für Java, Entwickeln Sie Verbraucher mit gemeinsamem Durchsatz mit dem AWS SDK für Java, und Kinesis-Datenstreams erstellen und verwalten.
- Weitere Informationen zur Kinesis Client Library finden Sie unter Entwickeln Sie KCL 1.x-Verbraucher.
- Weitere Informationen zur Optimierung Ihrer Anwendung finden Sie unter Optimieren Sie die Verbraucher von Amazon Kinesis Data Streams.

Tutorial: Analysieren Sie Aktiendaten in Echtzeit mit Amazon Managed Service für Apache Flink

Im Szenario dieses Tutorials werden Wertpapierdaten in einen Datenstrom geschrieben. Zudem wird eine einfache Amazon Managed Service für Apache Flink-Anwendung erstellt, die Berechnungen mit dem Stream durchführt. Sie lernen, wie Sie einen Stream von Datensätzen an Kinesis Data Streams senden und eine Anwendung implementieren, die die Datensätze nahezu in Echtzeit verarbeitet und verarbeitet.

Mit Amazon Managed Service für Apache Flink können Sie Java oder Scala verwenden, um Streaming-Daten zu verarbeiten und zu analysieren. Mit diesem Service können Sie Java- oder Scala-Code für Streaming-Quellen erstellen und ausführen, um Zeitreihenanalysen durchzuführen, Echtzeit-Dashboards zu füttern und Echtzeit-Metriken zu erstellen.

Sie können Flink-Anwendungen in Managed Service für Apache Flink mithilfe von Open-Source-Bibliotheken erstellen, die auf Apache Flink basieren. Apache Flink ist ein beliebtes Framework und eine verteilte Engine zum Verarbeiten von Datenströmen.

> **Important**
>
> Nachdem Sie zwei Datenstreams und eine Anwendung erstellt haben, fallen für Ihr Konto geringe Gebühren für Kinesis Data Streams und Managed Service für die Nutzung von Apache Flink an, da diese nicht für das AWS kostenlose Kontingent in Frage kommen. Wenn Sie mit dieser Anwendung fertig sind, löschen Sie Ihre AWS Ressourcen, damit keine Gebühren mehr anfallen.

Der Code greift nicht auf tatsächliche Wertpapierdaten zu, sondern simuliert nur deren Strom. Dazu werden zufällige Wertpapierdaten erzeugt. Wenn Sie Zugriff auf einen Echtzeit-Stream von Wertpapierdaten haben, möchten Sie vermutlich nützliche, zeitnahe Statistiken aus den Stream-Daten erzeugen. Sie können beispielsweise eine Zeitfensteranalyse durchführen, um festzustellen, welche Aktie in den letzten 5 Minuten am häufigsten erworben wurde. Oder Sie möchten im Falle eines zu großen Verkaufsauftrags (d. h. zu viele Anteile) benachrichtigt werden. Der Code in diesem Tutorial kann erweitert werden, um solche Funktionen bereitzustellen.

Bei den gezeigten Beispielen wird die Region USA West (Oregon) verwendet. Sie funktionieren aber auch für alle anderen AWS -Regionen, die Managed Service für Apache Flink unterstützen.

Aufgaben

- Voraussetzungen für das Abschließen der Übungen
- Richten Sie ein AWS Konto ein und erstellen Sie einen Administratorbenutzer
- Richten Sie das AWS Command Line Interface ()AWS CLI ein
- Erstellen Sie eine Managed Service for Apache Flink-Anwendung und führen Sie sie aus

Voraussetzungen für das Abschließen der Übungen

Zur Durchführung der Schritte in dieser Anleitung benötigen Sie Folgendes:

- Java Development Kit (JDK), Version 8. Legen Sie die `JAVA_HOME` Umgebungsvariable so fest, dass sie auf Ihren JDK-Installationsspeicherort weist.
- Wir empfehlen die Verwendung einer Entwicklungsumgebung (wie Eclipse Java Neon oder IntelliJ Idea), um Ihre Anwendung zu entwickeln und zu kompilieren.
- Git-Client. Installieren Sie den Git-Client, wenn Sie dies noch nicht getan haben.
- Apache Maven-Compiler-Plugin. Maven muss sich in Ihrem Arbeitspfad befinden. Zum Testen Ihrer Apache Maven-Installation geben Sie Folgendes ein:

```
$ mvn -version
```

Um zu beginnen, gehen Sie zu Richten Sie ein AWS Konto ein und erstellen Sie einen Administratorbenutzer.

Richten Sie ein AWS Konto ein und erstellen Sie einen Administratorbenutzer

Bevor Sie Amazon Managed Service für Apache Flink zum ersten Mal verwenden, führen Sie die folgenden Aufgaben aus:

1. Melden Sie sich an für AWS
2. Erstellen eines IAM-Benutzers

Melden Sie sich an für AWS

Wenn Sie sich für Amazon Web Services (AWS) registrieren, wird Ihr AWS Konto automatisch für alle Dienste angemeldet AWS, einschließlich Amazon Managed Service für Apache Flink. Berechnet werden Ihnen aber nur die Services, die Sie nutzen.

Mit Managed Service für Apache Flink zahlen Sie nur für die Ressourcen, die Sie wirklich nutzen. Wenn Sie ein neuer AWS -Kunde sind, können Sie kostenlos mit Managed Service für Apache Flink beginnen. Weitere Informationen finden Sie unter Kostenloses Kontingent für AWS.

Wenn Sie bereits ein AWS Konto haben, fahren Sie mit der nächsten Aufgabe fort. Wenn Sie noch kein AWS -Konto haben, führen Sie die folgenden Schritte aus, um ein Konto zu erstellen.

Um ein AWS Konto zu erstellen

1. Öffnen Sie https://portal.aws.amazon.com/billing/die Anmeldung.
2. Folgen Sie den Online-Anweisungen.

   Während der Anmeldung erhalten Sie einen Telefonanruf oder eine Textnachricht und müssen einen Verifizierungscode über die Telefontasten eingeben.

   Wenn Sie sich für eine anmelden AWS-Konto, Root-Benutzer des AWS-Kontoswird eine erstellt. Der Root-Benutzer hat Zugriff auf alle AWS-Services und Ressourcen des Kontos. Als bewährte Sicherheitsmethode weisen Sie einem Benutzer Administratorzugriff zu und verwenden Sie nur den Root-Benutzer, um Aufgaben auszuführen, die Root-Benutzerzugriff erfordern.

Notieren Sie sich Ihre AWS Konto-ID, da Sie sie für die nächste Aufgabe benötigen.

Erstellen eines IAM-Benutzers

Dienste in AWS, wie Amazon Managed Service für Apache Flink, erfordern, dass Sie beim Zugriff auf sie Anmeldeinformationen angeben. Der Service kann feststellen, ob Sie über die Berechtigung für den Zugriff auf die Ressourcen im Besitz dieses Service verfügen. Das AWS-Managementkonsole erfordert, dass Sie Ihr Passwort eingeben.

Sie können Zugangsschlüssel für Ihr AWS Konto erstellen, um auf die AWS Command Line Interface (AWS CLI) oder API zuzugreifen. Wir empfehlen jedoch nicht, dass Sie AWS mit den Anmeldeinformationen für Ihr AWS Konto darauf zugreifen. Stattdessen empfehlen wir Ihnen, AWS Identity and Access Management (IAM) zu verwenden. Erstellen Sie einen IAM-Benutzer und fügen Sie den Benutzer zu einer IAM-Gruppe mit Administrator-Berechtigungen hinzu. Anschließend

gewähren Sie dem von Ihnen erstellten IAM-Benutzer administrative Berechtigungen. Danach können Sie mithilfe einer speziellen URL und der Anmeldeinformationen des IAM-Benutzers auf AWS zugreifen.

Wenn Sie sich für registriert haben AWS, aber noch keinen IAM-Benutzer für sich selbst erstellt haben, können Sie einen mit der IAM-Konsole erstellen.

Für die Erste-Schritte-Übungen in diesem Handbuch wird davon ausgegangen, dass Sie einen Benutzer (`adminuser`) mit Administratorrechten haben. Befolgen Sie die Schritte zum Einrichten des `adminuser` in Ihrem Konto.

So erstellen Sie eine Gruppe für Administratoren:

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 Navigationsbereich Groups (Gruppen) und dann Create New Group (Neue Gruppe erstellen) aus.
3. Geben Sie für Group Name (Gruppenname) einen Namen für die Gruppe ein, z. B. **`Administrators`**. Wählen Sie dann Next Step (Nächster Schritt) aus.
4. Aktivieren Sie in der Liste der Richtlinien das Kontrollkästchen neben der AdministratorAccessRichtlinie. Über das Menü Filter (Filtern) und das Feld Search (Suchen) können Sie die Liste filtern.
5. Wählen Sie Next Step (Nächster Schritt) und anschließend Create Group (Gruppe erstellen) aus.

Ihre neue Gruppe wird unter Group Name aufgeführt.

Zum Erstellen eines IAM-Benutzers für sich selbst, fügen Sie ihn der Administratorengruppe hinzu und erstellen Sie ein Passwort

1. Wählen Sie im Navigationsbereich Users (Benutzer) und dann Add User (Benutzer hinzufügen) aus.
2. Geben Sie im Feld Benutzername einen Benutzernamen ein.
3. Wählen Sie die beiden Optionen Programmgesteuerter Zugriff und Zugriff auf die AWS - Managementkonsole aus.
4. Wählen Sie Weiter: Berechtigungen aus.
5. Aktivieren Sie das Kontrollkästchen neben der Administratorengruppe. Wählen Sie dann Next: Review aus.

6. Wählen Sie Create user (Benutzer erstellen) aus.

So melden Sie sich als neuer IAM-Benutzer an

1. Melden Sie sich von der ab AWS-Managementkonsole.
2. Verwenden Sie das folgende URL-Format zum Anmelden bei der Konsole:

   `https://aws_account_number.signin.aws.amazon.com/console/`

   Das *aws_account_number* ist Ihre AWS Konto-ID ohne Bindestriche. Wenn Ihre AWS Konto-ID beispielsweise 1234-5678-9012 lautet, ersetzen Sie sie durch. *aws_account_number* **123456789012** Informationen dazu, wie Sie Ihre Kontonummer finden, finden Sie unter Ihre AWS Konto-ID und ihr Alias im IAM-Benutzerhandbuch.
3. Geben Sie den IAM-Benutzernamen und das von Ihnen soeben erstellte Passwort ein. Wenn Sie angemeldet sind, wird in der Navigationsleiste *your_user_name* @ *your_aws_account_id* angezeigt.

> **Note**
>
> Wenn Sie nicht möchten, dass die URL für Ihre Anmeldeseite Ihre AWS Konto-ID enthält, können Sie einen Kontoalias erstellen.

So erstellen oder entfernen Sie einen Konto-Alias

1. Öffnen Sie unter https://console.aws.amazon.com/iam/ die IAM-Konsole.
2. Wählen Sie im Navigationsbereich Dashboard aus.
3. Suchen Sie den Anmeldelink für IAM-Benutzer.
4. Um einen Alias zu erstellen, klicken Sie auf Anpassen. Geben Sie den gewünschten Namen für den Alias ein und wählen Sie Yes, Create (Ja, erstellen) aus.
5. Um den Alias zu löschen, wählen Sie Customize (Anpassen) und dann Yes, Delete (Ja, löschen). Die Anmelde-URL verwendet wieder Ihre AWS Konto-ID.

Nach dem Erstellen eines Konto-Alias verwenden Sie die folgende URL, um sich anzumelden:

`https://your_account_alias.signin.aws.amazon.com/console/`

Um den Anmeldelink der IAM-Benutzer Ihres Kontos zu verifizieren, öffnen Sie die IAM-Konsole und prüfen dies im Dashboard unter IAM users sign-in link.

Weitere Informationen zu IAM finden Sie unter:

- AWS Identity and Access Management (ICH BIN)
- Erste Schritte mit IAM
- IAM Benutzerhandbuch

Nächster Schritt

Richten Sie das AWS Command Line Interface ()AWS CLI ein

Richten Sie das AWS Command Line Interface ()AWS CLI ein

In diesem Schritt laden Sie den herunter und konfigurieren ihn für AWS CLI die Verwendung mit Amazon Managed Service für Apache Flink.

> **Note**
>
> Bei allen Erste-Schritte-Übungen in diesem Handbuch wird davon ausgegangen, dass Sie in Ihrem Konto Administrator-Anmeldeinformationen (`adminuser`) verwenden, um die Operationen auszuführen.

> **Note**
>
> Wenn Sie den bereits AWS CLI installiert haben, müssen Sie möglicherweise ein Upgrade durchführen, um die neuesten Funktionen zu erhalten. Weitere Informationen finden Sie im AWS Command Line Interface Benutzerhandbuch unter Installation der AWS Befehlszeilenschnittstelle. Führen Sie den folgenden Befehl aus AWS CLI, um die Version von zu überprüfen:
>
> ```
> aws --version
> ```
>
> Für die Übungen in diesem Tutorial ist die folgende AWS CLI Version oder höher erforderlich:

```
aws-cli/1.16.63
```

Um das einzurichten AWS CLI

1. Herunterladen und Konfigurieren von AWS CLI. Eine Anleitung finden Sie unter den folgenden Themen im AWS Command Line Interface -Benutzerhandbuch:

   - Installieren des AWS Command Line Interface
   - Konfigurieren von AWS CLI

2. Fügen Sie der AWS CLI Konfigurationsdatei ein benanntes Profil für den Administratorbenutzer hinzu. Sie verwenden dieses Profil, wenn Sie die AWS CLI Befehle ausführen. Weitere Informationen zu benannten Profilen finden Sie unter Benannte Profile im AWS Command Line Interface Benutzerhandbuch.

   ```
   [profile adminuser]
   aws_access_key_id = adminuser access key ID
   aws_secret_access_key = adminuser secret access key
   region = aws-region
   ```

   Eine Liste der verfügbaren AWS Regionen finden Sie unter AWS Regionen und Endpunkte in der Allgemeine Amazon Web Services-Referenz.

3. Überprüfen Sie die Einrichtung, indem Sie die folgenden Hilfebefehle in die Befehlszeile eingeben:

   ```
   aws help
   ```

Nachdem Sie ein AWS Konto eingerichtet haben AWS CLI, können Sie die nächste Übung ausprobieren, in der Sie eine Beispielanwendung konfigurieren und das end-to-end Setup testen.

Nächster Schritt

Erstellen Sie eine Managed Service for Apache Flink-Anwendung und führen Sie sie aus

Erstellen Sie eine Managed Service for Apache Flink-Anwendung und führen Sie sie aus

In dieser Übung erstellen Sie eine Anwendung von Managed Service für Apache Flink mit Datenströmen als Quelle und Senke.

Dieser Abschnitt enthält die folgenden Schritte:

- Erstellen Sie zwei Amazon Kinesis Kinesis-Datenstreams
- Schreiben Sie Beispieldatensätze in den Eingabe-Stream
- Laden Sie den Apache Flink-Streaming-Java-Code herunter und untersuchen Sie ihn
- Kompilieren Sie den Anwendungscode
- Laden Sie den Apache Flink-Streaming-Java-Code hoch
- Erstellen Sie die Anwendung Managed Service for Apache Flink und führen Sie sie aus

Erstellen Sie zwei Amazon Kinesis Kinesis-Datenstreams

Bevor Sie für diese Übung einen Amazon Managed Service für Apache Flink erstellen, erstellen Sie zwei Kinesis-Datenstreams (`ExampleInputStream`und`ExampleOutputStream`). Ihre Anwendung verwendet diese Streams für die Quell- und Ziel-Streams der Anwendung.

Sie können diese Streams mithilfe der Amazon-Kinesis-Konsole oder des folgenden AWS CLI -Befehls erstellen. Detaillierte Konsolenanweisungen finden Sie unter Erstellen und Aktualisieren von Daten-Streams.

So erstellen Sie die Daten-Streams (AWS CLI)

1. Verwenden Sie den folgenden Amazon Kinesis `create-stream` AWS CLI Kinesis-Befehl, um den ersten Stream (`ExampleInputStream`) zu erstellen.

   ```
   $ aws kinesis create-stream \
   --stream-name ExampleInputStream \
   --shard-count 1 \
   --region us-west-2 \
   --profile adminuser
   ```

2. Um den zweiten Stream zu erstellen, den die Anwendung zum Schreiben der Ausgabe verwendet, führen Sie denselben Befehl aus und ändern den Stream-Namen in `ExampleOutputStream`.

```
$ aws kinesis create-stream \
--stream-name ExampleOutputStream \
--shard-count 1 \
--region us-west-2 \
--profile adminuser
```

Schreiben Sie Beispieldatensätze in den Eingabe-Stream

In diesem Abschnitt verwenden Sie ein Python-Skript zum Schreiben von Datensätzen in den Stream für die zu verarbeitende Anwendung.

> **Note**
>
> Dieser Abschnitt erfordert AWS SDK für Python (Boto).

1. Erstellen Sie eine Datei `stock.py` mit dem folgenden Inhalt:

```
import datetime
import json
import random
import boto3

STREAM_NAME = "ExampleInputStream"


def get_data():
    return {
        "EVENT_TIME": datetime.datetime.now().isoformat(),
        "TICKER": random.choice(["AAPL", "AMZN", "MSFT", "INTC", "TBV"]),
        "PRICE": round(random.random() * 100, 2),
    }


def generate(stream_name, kinesis_client):
    while True:
        data = get_data()
        print(data)
        kinesis_client.put_record(
```

```
            StreamName=stream_name, Data=json.dumps(data),
 PartitionKey="partitionkey"
        )


if __name__ == "__main__":
    generate(STREAM_NAME, boto3.client("kinesis"))
```

2. Im weiteren Verlauf des Tutorials führen Sie das `stock.py`-Skript zum Senden von Daten an die Anwendung aus.

```
$ python stock.py
```

Laden Sie den Apache Flink-Streaming-Java-Code herunter und untersuchen Sie ihn

Der Java-Anwendungscode für diese Beispiele ist verfügbar unter GitHub. Zum Herunterladen des Anwendungscodes gehen Sie wie folgt vor:

1. Klonen Sie das Remote-Repository mit dem folgenden Befehl:

```
git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-java-
examples.git
```

2. Navigieren Sie zum `GettingStarted` Verzeichnis .

Der Anwendungscode befindet sich in den Dateien `CloudWatchLogSink.java` und `CustomSinkStreamingJob.java`. Beachten Sie Folgendes zum Anwendungscode:

- Die Anwendung verwendet eine Kinesis-Quelle zum Lesen aus dem Quell-Stream. Der folgende Codeausschnitt erstellt die Kinesis-Senke:

```
return env.addSource(new FlinkKinesisConsumer<>(inputStreamName,
                new SimpleStringSchema(), inputProperties));
```

Kompilieren Sie den Anwendungscode

In diesem Abschnitt verwenden Sie den Apache Maven-Compiler zum Erstellen des Java-Codes für die Anwendung. Weitere Informationen zum Installieren von Apache Maven und des Java Development Kit (JDK) finden Sie unter Voraussetzungen für das Abschließen der Übungen.

Ihre Java-Anwendung erfordert die folgenden Komponenten:

- Eine Projektobjektmodell (pom.xml)-Datei. Diese Datei enthält Informationen zur Konfiguration und zu den Abhängigkeiten der Anwendung, einschließlich der Amazon Managed Service for Apache Flink-Bibliotheken.
- Eine `main`-Methode, die die Logik der Anwendung enthält.

> **Note**
>
> Um den Kinesis-Konnektor für die folgende Anwendung zu verwenden, müssen Sie den Quellcode für den Konnektor herunterladen und ihn wie in der Apache Flink-Dokumentation beschrieben erstellen.

So erstellen und kompilieren Sie den Anwendungscode

1. Erstellen Sie eine Java/Maven Anwendung in Ihrer Entwicklungsumgebung. Weitere Informationen zum Erstellen einer Anwendung finden Sie in der Dokumentation für Ihre Entwicklungsumgebung:
   - Erstellen Sie Ihr erstes Java-Projekt (Eclipse Java Neon)
   - Erstellen, Ausführen und Packen Ihrer ersten Java-Anwendung (IntelliJ Idea)
2. Verwenden Sie den folgenden Code für eine Datei mit dem Namen `StreamingJob.java`.

```
package com.amazonaws.services.kinesisanalytics;

import com.amazonaws.services.kinesisanalytics.runtime.KinesisAnalyticsRuntime;
import org.apache.flink.api.common.serialization.SimpleStringSchema;
import org.apache.flink.streaming.api.datastream.DataStream;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer;
import org.apache.flink.streaming.connectors.kinesis.FlinkKinesisProducer;
```

```
import
 org.apache.flink.streaming.connectors.kinesis.config.ConsumerConfigConstants;

import java.io.IOException;
import java.util.Map;
import java.util.Properties;

public class StreamingJob {

    private static final String region = "us-east-1";
    private static final String inputStreamName = "ExampleInputStream";
    private static final String outputStreamName = "ExampleOutputStream";

    private static DataStream<String>
 createSourceFromStaticConfig(StreamExecutionEnvironment env) {
        Properties inputProperties = new Properties();
        inputProperties.setProperty(ConsumerConfigConstants.AWS_REGION, region);

 inputProperties.setProperty(ConsumerConfigConstants.STREAM_INITIAL_POSITION,
 "LATEST");

        return env.addSource(new FlinkKinesisConsumer<>(inputStreamName, new
 SimpleStringSchema(), inputProperties));
    }

    private static DataStream<String>
 createSourceFromApplicationProperties(StreamExecutionEnvironment env)
            throws IOException {
        Map<String, Properties> applicationProperties =
 KinesisAnalyticsRuntime.getApplicationProperties();
        return env.addSource(new FlinkKinesisConsumer<>(inputStreamName, new
 SimpleStringSchema(),
                applicationProperties.get("ConsumerConfigProperties")));
    }

    private static FlinkKinesisProducer<String> createSinkFromStaticConfig() {
        Properties outputProperties = new Properties();
        outputProperties.setProperty(ConsumerConfigConstants.AWS_REGION, region);
        outputProperties.setProperty("AggregationEnabled", "false");

        FlinkKinesisProducer<String> sink = new FlinkKinesisProducer<>(new
 SimpleStringSchema(), outputProperties);
        sink.setDefaultStream(outputStreamName);
        sink.setDefaultPartition("0");
```

```
        return sink;
    }

    private static FlinkKinesisProducer<String>
 createSinkFromApplicationProperties() throws IOException {
        Map<String, Properties> applicationProperties =
 KinesisAnalyticsRuntime.getApplicationProperties();
        FlinkKinesisProducer<String> sink = new FlinkKinesisProducer<>(new
 SimpleStringSchema(),
                applicationProperties.get("ProducerConfigProperties"));

        sink.setDefaultStream(outputStreamName);
        sink.setDefaultPartition("0");
        return sink;
    }

    public static void main(String[] args) throws Exception {
        // set up the streaming execution environment
        final StreamExecutionEnvironment env =
 StreamExecutionEnvironment.getExecutionEnvironment();

        /*
         * if you would like to use runtime configuration properties, uncomment the
         * lines below
         * DataStream<String> input = createSourceFromApplicationProperties(env);
         */

        DataStream<String> input = createSourceFromStaticConfig(env);

        /*
         * if you would like to use runtime configuration properties, uncomment the
         * lines below
         * input.addSink(createSinkFromApplicationProperties())
         */

        input.addSink(createSinkFromStaticConfig());

        env.execute("Flink Streaming Java API Skeleton");
    }
}
```

Beachten Sie die folgenden Informationen zum vorherigen Codebeispiel:

- Diese Datei enthält die `main`-Methode, die die Funktionalität der Anwendung definiert.
- Ihre Anwendung erstellt Quell- und Senkenkonnektoren für den Zugriff auf externe Ressourcen, indem ein `StreamExecutionEnvironment`-Objekt verwendet wird.
- Die Anwendung erstellt Quell- und Senkenkonnektoren mit statischen Eigenschaften. Zum Verwenden dynamischer Anwendungseigenschaften verwenden Sie die Methoden `createSourceFromApplicationProperties` und `createSinkFromApplicationProperties`, um die Konnektoren zu erstellen. Diese Methoden lesen die Eigenschaften der Anwendung zum Konfigurieren der Konnektoren.

3. Zum Verwenden Ihres Anwendungscodes kompilieren und packen Sie ihn in eine JAR-Datei. Sie können Ihren Code auf zwei Arten kompilieren und packen:

   - Verwenden Sie das Befehlszeilen-Maven-Tool. Erstellen Sie Ihre JAR-Datei, indem Sie den folgenden Befehl in dem Verzeichnis ausführen, das die `pom.xml`-Datei enthält:

     ```
     mvn package
     ```

   - Verwenden Sie Ihre Entwicklungsumgebung. Weitere Informationen finden Sie in der Dokumentation Ihrer Entwicklungsumgebung.

   Sie können Ihr Paket als JAR-Datei hochladen oder komprimieren und als ZIP-Datei hochladen. Wenn Sie Ihre Anwendung mit dem erstellen AWS CLI, geben Sie Ihren Codeinhaltstyp (JAR oder ZIP) an.
4. Wenn während der Erstellung Fehler aufgetreten sind, überprüfen Sie, ob Ihre `JAVA_HOME`-Umgebungsvariable richtig eingestellt ist.

Wenn die Anwendung erfolgreich kompiliert wurde, wird die folgende Datei erstellt:

`target/java-getting-started-1.0.jar`

Laden Sie den Apache Flink-Streaming-Java-Code hoch

In diesem Abschnitt erstellen Sie einen Amazon Simple Storage Service (Amazon S3)-Bucket und laden Ihren Anwendungscode hoch.

So laden Sie den Anwendungscode hoch

1. Öffnen Sie die Amazon S3 S3-Konsole unter https://console.aws.amazon.com/s3/.

2. Wählen Sie Create Bucket (Bucket erstellen) aus.
3. Geben Sie **`ka-app-code-<username>`** im Feld Bucket-Name ein. Fügen Sie dem Bucket-Namen ein Suffix hinzu, wie z. B. Ihren Benutzernamen, damit er global eindeutig ist. Wählen Sie Weiter aus.
4. Lassen Sie im Schritt Optionen konfigurieren die Einstellungen unverändert und klicken Sie auf Weiter.
5. Lassen Sie im Schritt Berechtigungen festlegen die Einstellungen unverändert und klicken Sie auf Weiter.
6. Wählen Sie Create Bucket (Bucket erstellen) aus.
7. Wählen Sie in der Amazon S3 S3-Konsole den `<username>` Bucket ka-app-code- und wählen Sie Upload aus.
8. Klicken Sie im Schritt Auswählen von Dateien auf Hinzufügen von Dateien. Navigieren Sie zu der `java-getting-started-1.0.jar`Datei, die Sie im vorherigen Schritt erstellt haben. Wählen Sie Weiter aus.
9. Lassen Sie im Schritt Berechtigungen festlegen die Einstellungen unverändert. Wählen Sie Weiter aus.
10. Lassen Sie im Schritt Eigenschaften festlegen die Einstellungen unverändert. Klicken Sie auf Upload.

Ihr Anwendungscode ist jetzt in einem Amazon-S3-Bucket gespeichert, in dem Ihre Anwendung darauf zugreifen kann.

Erstellen Sie die Anwendung Managed Service for Apache Flink und führen Sie sie aus

Sie können eine Anwendung von Managed Service für Apache Flink entweder über die Konsole oder AWS CLI erstellen und ausführen.

> **Note**
>
> Wenn Sie die Anwendung mithilfe der Konsole erstellen, werden Ihre AWS Identity and Access Management (IAM) und Amazon CloudWatch Logs-Ressourcen für Sie erstellt. Wenn Sie die Anwendung mithilfe von erstellen AWS CLI, erstellen Sie diese Ressourcen separat.

Themen

- Erstellen Sie die Anwendung und führen Sie sie aus (Konsole)
- Erstellen Sie die Anwendung und führen Sie sie aus (AWS CLI)

Erstellen Sie die Anwendung und führen Sie sie aus (Konsole)

Befolgen Sie diese Schritte, um die Anwendung über die Konsole zu erstellen, zu konfigurieren, zu aktualisieren und auszuführen.

Erstellen der Anwendung

1. Öffnen Sie die Kinesis-Konsole unter https://console.aws.amazon.com/kinesis.
2. Wählen Sie auf dem Amazon-Kinesis-Dashboard die Option Create analytics application (Analyseanwendung erstellen) aus.
3. Geben Sie auf der Seite Kinesis Analytics – Anwendung erstellen die Anwendungsdetails wie folgt an:
   - Geben Sie als Anwendungsname ein **`MyApplication`**.
   - Geben Sie für Beschreibung den Text **`My java test app`** ein.
   - Wählen Sie für Runtime (Laufzeit) die Option Apache Flink 1.6 aus.
4. Wählen Sie für Zugriffsberechtigungen die Option Erstellen / Aktualisieren Sie IAM-Rolle **`kinesis-analytics-MyApplication-us-west-2`** aus.
5. Wählen Sie Create application aus.

> **Note**
>
> Wenn Sie mithilfe der Konsole eine Amazon Managed Service for Apache Flink-Anwendung erstellen, haben Sie die Möglichkeit, eine IAM-Rolle und -Richtlinie für Ihre Anwendung erstellen zu lassen. Ihre Anwendung verwendet diese Rolle und Richtlinie für den Zugriff auf ihre abhängigen Ressourcen. Diese IAM-Ressourcen werden unter Verwendung Ihres Anwendungsnamens und der Region wie folgt benannt:
>
> - Richtlinie: `kinesis-analytics-service-`*`MyApplication`*`-`*`us-west-2`*
> - Rolle: `kinesis-analytics-`*`MyApplication`*`-`*`us-west-2`*

Bearbeiten Sie die IAM-Richtlinie

Bearbeiten Sie die IAM-Richtlinie zum Hinzufügen von Berechtigungen für den Zugriff auf die Kinesis-Datenströme.

1. Öffnen Sie unter https://console.aws.amazon.com/iam/ die IAM-Konsole.
2. Wählen Sie Policies (Richtlinien). Wählen Sie die **`kinesis-analytics-service-MyApplication-us-west-2`**-Richtlinie aus, die die Konsole im vorherigen Abschnitt für Sie erstellt hat.
3. Wählen Sie auf der Seite Summary (Übersicht) die Option Edit policy (Richtlinie bearbeiten) aus. Wählen Sie den Tab JSON.
4. Fügen Sie den markierten Abschnitt der folgenden Beispielrichtlinie der Richtlinie hinzu. Ersetzen Sie das Beispielkonto IDs (*012345678901*) durch Ihre Konto-ID.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:s3:::ka-app-code-username/java-getting-
started-1.0.jar"
            ]
        },
        {
            "Sid": "ListCloudwatchLogGroups",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*"
            ]
```

```
        },
        {
            "Sid": "ListCloudwatchLogStreams",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:*"
            ]
        },
        {
            "Sid": "PutCloudwatchLogs",
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteOutputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
        }
    ]
}
```

Konfigurieren Sie die Anwendung

1. Wählen Sie auf der MyApplicationSeite Configure aus.

2. Klicken Sie auf der Seite Configure application (Anwendung konfigurieren) auf die Option Code location (Codespeicherort):
   - Geben Sie für Amazon-S3-Bucket **ka-app-code-*<username>*** ein.
   - Geben Sie als Pfad zum Amazon-S3-Objekt den Wert **java-getting-started-1.0.jar** ein.
3. Wählen Sie unter Zugriff auf Anwendungsressourcen für Zugriffsberechtigungen die Option IAM-Rolle **kinesis-analytics-MyApplication-us-west-2** erstellen/aktualisieren aus.
4. Geben Sie unter Eigenschaften für Gruppen-ID den Text **ProducerConfigProperties** ein.
5. Geben Sie die folgenden Eigenschaften und Werte der Anwendung ein:

Key (Schlüssel)	Value (Wert)
flink.inputstream.initpos	**LATEST**
aws:region	**us-west-2**
AggregationEnabled	**false**

6. Stellen Sie unter Überwachung sicher, dass die Ebene der Überwachungsmetriken auf Anwendung eingestellt ist.
7. Wählen Sie für die CloudWatch Protokollierung das Kontrollkästchen Aktivieren aus.
8. Wählen Sie Aktualisieren aus.

> **Note**
>
> Wenn Sie die CloudWatch Protokollierung aktivieren möchten, erstellt Managed Service for Apache Flink eine Protokollgruppe und einen Protokollstream für Sie. Die Namen dieser Ressourcen lauten wie folgt:
>
> - Protokollgruppe: `/aws/kinesis-analytics/MyApplication`
> - Protokollstream: `kinesis-analytics-log-stream`

Führen Sie die Anwendung aus.

1. Wählen Sie auf der MyApplicationSeite die Option Ausführen aus. Bestätigen Sie die Aktion.

2. Wenn die Anwendung ausgeführt wird, aktualisieren Sie die Seite. Die Konsole zeigt den Application graph (Anwendungs-Graph) an.

Beenden Sie die Anwendung

Wählen Sie auf der MyApplicationSeite Stopp aus. Bestätigen Sie die Aktion.

Aktualisieren der Anwendung

Mithilfe der Konsole können Sie Anwendungseinstellungen wie beispielsweise Anwendungseigenschaften, Überwachungseinstellungen und den Speicherort oder den Dateinamen der JAR-Anwendungsdatei aktualisieren. Außerdem können Sie die JAR-Anwendungsdatei erneut aus dem Amazon-S3-Bucket laden, wenn Sie den Anwendungscode aktualisieren müssen.

Wählen Sie auf der MyApplicationSeite Configure aus. Aktualisieren Sie die Anwendungseinstellungen und klicken Sie auf Aktualisieren.

Erstellen Sie die Anwendung und führen Sie sie aus (AWS CLI)

In diesem Abschnitt verwenden Sie die, AWS CLI um die Anwendung Managed Service for Apache Flink zu erstellen und auszuführen. Managed Service for Apache Flink verwendet den `kinesisanalyticsv2` AWS CLI Befehl, um Managed Service for Apache Flink-Anwendungen zu erstellen und mit ihnen zu interagieren.

Erstellen einer Berechtigungsrichtlinie

Zuerst erstellen Sie eine Berechtigungsrichtlinie mit zwei Anweisungen: eine, die Berechtigungen für die `read`-Aktion auf den Quell-Stream zulässt, und eine andere, die Berechtigungen für die `write`-Aktionen auf den Senken-Stream zulässt. Anschließend fügen Sie die Richtlinie an eine IAM-Rolle (die Sie im nächsten Abschnitt erstellen) an. Wenn Managed Service für Apache Flink also die Rolle übernimmt, verfügt der Service über die erforderlichen Berechtigungen zum Lesen aus dem Quell-Stream und zum Schreiben in den Senken-Stream.

Verwenden Sie den folgenden Code zum Erstellen der `KAReadSourceStreamWriteSinkStream`-Berechtigungsrichtlinie. Ersetzen Sie *`username`* durch den Benutzernamen, den Sie verwendet haben, um den Amazon-S3-Bucket zum Speichern des Anwendungscodes zu erstellen. Ersetzen Sie die Konto-ID in den Amazon-Ressourcennamen (ARNs) (*`012345678901`*) durch Ihre Konto-ID.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "S3",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": ["arn:aws:s3:::ka-app-code-username",
                "arn:aws:s3:::ka-app-code-username/*"
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteOutputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
        }
    ]
}
```

step-by-stepAnweisungen zum Erstellen einer Berechtigungsrichtlinie finden Sie unter Tutorial: Erstellen und Anhängen Ihrer ersten vom Kunden verwalteten Richtlinie im IAM-Benutzerhandbuch.

> **Note**
>
> Um auf andere AWS Dienste zuzugreifen, können Sie den AWS SDK für Java verwenden. Managed Service für Apache Flink setzt die vom SDK benötigten Anmeldeinformationen

automatisch auf die der IAM-Rolle für die Dienstausführung, die mit Ihrer Anwendung verknüpft ist. Es sind keine weiteren Schritte erforderlich.

Erstellen einer IAM-Rolle

In diesem Abschnitt erstellen Sie eine IAM-Rolle, von der Managed Service for Apache Flink annehmen kann, dass sie einen Quellstream liest und in den Sink-Stream schreibt.

Managed Service für Apache Flink kann ohne Berechtigungen nicht auf Ihren Stream zugreifen. Sie erteilen diese Berechtigungen über eine IAM-Rolle. Jeder IAM-Rolle sind zwei Richtlinien angefügt. Die Vertrauensrichtlinie erteilt Managed Service für Apache Flink die Berechtigung zum Übernehmen der Rolle und die Berechtigungsrichtlinie bestimmt, was Managed Service für Apache Flink nach Annahme der Rolle tun kann.

Sie können die Berechtigungsrichtlinie, die Sie im vorherigen Abschnitt erstellt haben, dieser Rolle anfügen.

So erstellen Sie eine IAM-Rolle

1. Öffnen Sie unter https://console.aws.amazon.com/iam/ die IAM-Konsole.
2. Wählen Sie im Navigationsbereich Roles (Rollen) und Create Role (Rolle erstellen) aus.
3. Wählen Sie unter Typ der vertrauenswürdigen Entität auswählen die Option AWS -Service aus. Wählen Sie unter Choose the service that will use this role (Wählen Sie den Service aus, der diese Rolle verwendet) die Option Kinesis aus. Wählen Sie unter Select your use case (Wählen Sie Ihren Anwendungsfall aus) die Option Kinesis Analytics aus.

   Wählen Sie Weiter: Berechtigungen aus.
4. Wählen Sie auf der Seite Attach permissions policies (Berechtigungsrichtlinien hinzufügen) Next: Review (Weiter: Überprüfen) aus. Sie fügen Berechtigungsrichtlinien an, nachdem Sie die Rolle erstellt haben.
5. Geben Sie auf der Seite Create role (Rolle erstellen) den Text **`KA-stream-rw-role`** für Role name (Rollenname) ein. Wählen Sie Rolle erstellen aus.

   Jetzt haben Sie eine neue IAM-Rolle mit dem Namen `KA-stream-rw-role` erstellt. Im nächsten Schritt aktualisieren Sie die Vertrauens- und Berechtigungsrichtlinien für die Rolle.
6. Fügen Sie die Berechtigungsrichtlinie der Rolle an.

> **Note**
>
> Für diese Übung übernimmt Managed Service für Apache Flink diese Rolle sowohl für das Lesen von Daten aus einem Kinesis-Datenstrom (Quelle) als auch zum Schreiben der Ausgabedaten in einen anderen Kinesis-Datenstrom. Daher fügen Sie die Richtlinie an, die Sie im vorherigen Schritt erstellt haben, the section called “Erstellen einer Berechtigungsrichtlinie”.

a. Wählen Sie auf der Seite Summary (Übersicht) die Registerkarte Permissions (Berechtigungen) aus.

b. Wählen Sie Attach Policies (Richtlinien anfügen) aus.

c. Geben Sie im Suchfeld **`KAReadSourceStreamWriteSinkStream`** (die Richtlinie, die Sie im vorhergehenden Abschnitt erstellt haben) ein.

d. Wählen Sie die KAReadInputStreamWriteOutputStream-Richtlinie und wählen Sie Richtlinie anhängen aus.

Sie haben nun die Service-Ausführungsrolle erstellt, die Ihre Anwendung für den Zugriff auf Ressourcen verwendet. Notieren Sie sich den ARN der neuen Rolle.

step-by-stepAnweisungen zum Erstellen einer Rolle finden Sie unter Erstellen einer IAM-Rolle (Konsole) im IAM-Benutzerhandbuch.

Erstellen Sie die Anwendung Managed Service für Apache Flink

1. Speichern Sie den folgenden JSON-Code in eine Datei mit dem Namen `create_request.json`. Ersetzen Sie den Beispiel-Rollen-ARN durch den ARN für die Rolle, die Sie zuvor erstellt haben. Ersetzen Sie das Bucket-ARN-Suffix (*`username`*) mit dem Suffix, das Sie im vorherigen Abschnitt gewählt haben. Ersetzen Sie die beispielhafte Konto-ID (*`012345678901`*) in der Service-Ausführungsrolle mit Ihrer Konto-ID.

```
{
    "ApplicationName": "test",
    "ApplicationDescription": "my java test app",
    "RuntimeEnvironment": "FLINK-1_6",
    "ServiceExecutionRole": "arn:aws:iam::012345678901:role/KA-stream-rw-role",
    "ApplicationConfiguration": {
```

```
        "ApplicationCodeConfiguration": {
            "CodeContent": {
                "S3ContentLocation": {
                    "BucketARN": "arn:aws:s3:::ka-app-code-username",
                    "FileKey": "java-getting-started-1.0.jar"
                }
            },
            "CodeContentType": "ZIPFILE"
        },
        "EnvironmentProperties":  {
         "PropertyGroups": [
            {
               "PropertyGroupId": "ProducerConfigProperties",
               "PropertyMap" : {
                    "flink.stream.initpos" : "LATEST",
                    "aws.region" : "us-west-2",
                    "AggregationEnabled" : "false"
               }
            },
            {
               "PropertyGroupId": "ConsumerConfigProperties",
               "PropertyMap" : {
                    "aws.region" : "us-west-2"
               }
            }
         ]
      }
    }
}
```

2. Führen Sie die `CreateApplication`-Aktion mit der vorherigen Anforderung zum Erstellen der Anwendung aus:

```
aws kinesisanalyticsv2 create-application --cli-input-json file://
create_request.json
```

Die Anwendung wird nun erstellt. Sie starten die Anwendung im nächsten Schritt.

Starten der Anwendung

In diesem Abschnitt verwenden Sie die `StartApplication`-Aktion, um die Anwendung zu starten.

So starten Sie die Anwendung

1. Speichern Sie den folgenden JSON-Code in eine Datei mit dem Namen `start_request.json`.

```
{
    "ApplicationName": "test",
    "RunConfiguration": {
        "ApplicationRestoreConfiguration": {
         "ApplicationRestoreType": "RESTORE_FROM_LATEST_SNAPSHOT"
         }
    }
}
```

2. Führen Sie die `StartApplication`-Aktion mit der vorherigen Anforderung zum Starten der Anwendung aus:

```
aws kinesisanalyticsv2 start-application --cli-input-json file://start_request.json
```

Die Anwendung wird jetzt ausgeführt. Sie können die Kennzahlen Managed Service for Apache Flink auf der CloudWatch Amazon-Konsole überprüfen, um sicherzustellen, dass die Anwendung funktioniert.

Stoppen der Anwendung

In diesem Abschnitt verwenden Sie die `StopApplication`-Aktion, um die Anwendung zu stoppen.

So stoppen Sie die Anwendung

1. Speichern Sie den folgenden JSON-Code in eine Datei mit dem Namen `stop_request.json`.

```
{"ApplicationName": "test"
}
```

2. Führen Sie die `StopApplication`-Aktion mit der folgenden Anforderung zum Stoppen der Anwendung aus:

```
aws kinesisanalyticsv2 stop-application --cli-input-json file://stop_request.json
```

Die Anwendung wird nun gestoppt.

Tutorial: Verwendung AWS Lambda mit Amazon Kinesis Data Streams

In dieser Anleitung erstellen Sie eine Lambda-Funktion, um Ereignisse aus einem Kinesis-Datenstrom zu nutzen. In diesem Beispielszenario schreibt eine benutzerdefinierte Anwendung Datensätze in einen Kinesis-Datenstrom. AWS Lambda fragt dann diesen Datenstrom ab und ruft, wenn er neue Datensätze erkennt, Ihre Lambda-Funktion auf. AWS Lambda führt dann die Lambda-Funktion aus, indem es die Ausführungsrolle annimmt, die Sie bei der Erstellung der Lambda-Funktion angegeben haben.

Eine ausführliche schrittweise Anleitung finden Sie unter Tutorial: Verwenden von AWS Lambda mit Amazon Kinesis.

> **Note**
>
> In diesem Tutorial wird davon ausgegangen, dass Sie über einige Kenntnisse der grundlegenden Lambda-Operationen und der AWS Lambda Konsole verfügen. Falls Sie dies noch nicht getan haben, folgen Sie den Anweisungen unter Erste Schritte mit AWS Lambda, um Ihre erste Lambda-Funktion zu erstellen.

Verwenden Sie die AWS Streaming-Datenlösung für Amazon Kinesis

Die AWS Streaming-Datenlösung für Amazon Kinesis konfiguriert automatisch die AWS Services, die für die einfache Erfassung, Speicherung, Verarbeitung und Bereitstellung von Streaming-Daten erforderlich sind. Die Lösung bietet mehrere Optionen zur Lösung von Anwendungsfällen für Streaming-Daten, die mehrere AWS Dienste nutzen, darunter Kinesis Data Streams AWS Lambda, Amazon API Gateway und Amazon Managed Service für Apache Flink.

Jede Lösung enthält die folgenden Komponenten:

- Ein CloudFormation Paket zur Bereitstellung des vollständigen Beispiels.
- Ein CloudWatch Dashboard zur Anzeige von Anwendungsmetriken.
- CloudWatch Alarme zu den relevantesten Anwendungsmetriken.

- Alle erforderlichen IAM-Rollen und -Richtlinien.

Die Lösung finden Sie hier: Streaming-Datenlösung für Amazon Kinesis

Kinesis-Datenstreams erstellen und verwalten

Amazon Kinesis Data Streams übernimmt eine große Menge von Daten in Echtzeit, speichert diese dauerhaft und stellt sie für den Verbrauch zur Verfügung. Die von Kinesis Data Streams gespeicherte Dateneinheit wird als Datensatz bezeichnet. Ein Datenstream repräsentiert eine Gruppe von Datensätzen. Die Datensätze in einem Daten-Stream werden auf Shards verteilt.

Ein Shard hat eine Sequenz von Datensätzen in einem Stream. Sie dient als Basisdurchsatzeinheit eines Kinesis Data Streams. Ein Shard unterstützt 1 MB/s und 1000 Datensätze pro Sekunde für Schreibvorgänge und 2 MB/s für Lesevorgänge, sowohl im On-Demand-Modus als auch im Bereitstellungsmodus. Die Shard-Limits sorgen für eine vorhersehbare Leistung und erleichtern so die Entwicklung und den Betrieb eines äußerst zuverlässigen Datenstreaming-Workflows.

In diesem Abschnitt erfahren Sie, wie Sie den Kapazitätsmodus für den Stream festlegen und wie Sie einen Stream entweder mit oder erstellen. AWS-Managementkonsole APIs Anschließend können Sie weitere Aktionen für den Stream ausführen.

Themen

- Wählen Sie den richtigen Modus zum Streamen
- Erstellen Sie einen Stream mit dem AWS-Managementkonsole
- Erstellen Sie einen Stream mit dem APIs
- Aktualisieren Sie einen Stream
- Auflisten von Streams
- Shards auflisten
- Löschen Sie einen Stream
- Einen Stream erneut teilen
- Ändern Sie den Aufbewahrungszeitraum für Daten
- Kennzeichnen Sie Ihre Amazon Kinesis Data Streams Streams-Ressourcen
- Umgang mit großen Datensätzen
- Führen Sie Resilienztests durch mit AWS Fault Injection Service

Wählen Sie den richtigen Modus zum Streamen

In den folgenden Themen wird erklärt, wie Sie den besten Modus für Ihre Anwendung auswählen und bei Bedarf zwischen den Modi wechseln können.

Themen

- Was sind die verschiedenen Modi in Kinesis Data Streams?
- Funktionen und Anwendungsfälle im On-Demand-Standardmodus
- Funktionen und Anwendungsfälle des Advantage-Modus auf Abruf
- Funktionen und Anwendungsfälle im Bereitstellungsmodus
- Wechseln Sie zwischen den Modi

Was sind die verschiedenen Modi in Kinesis Data Streams?

Ein Modus bestimmt, wie die Kapazität eines Datenstroms verwaltet wird und wie Ihnen die Nutzung Ihres Datenstroms in Rechnung gestellt wird. In Amazon Kinesis Data Streams können Sie On-Demand-Standard, On-Demand-Advantage und Bereitgestellt als Modus für Ihre Datenstreams wählen.

- On-Demand-Standard — Datenstreams mit einem On-Demand-Modus erfordern keine Kapazitätsplanung und werden automatisch skaliert, um Gigabytes an Schreib- und Lesedurchsatz pro Minute zu bewältigen. Im On-Demand-Modus verwaltet Kinesis Data Streams die Shards automatisch, um den erforderlichen Durchsatz bereitzustellen.
- Vorteil auf Abruf — Ein Modus auf Kontoebene, der mehr Funktionen ermöglicht und eine einfachere Preisstruktur für On-Demand-Streams bietet. In diesem Modus können Sie die Schreibdurchsatzkapazität eines Streams jederzeit proaktiv erhöhen. Bei der Preisgestaltung gibt es keine feste Gebühr mehr pro Stream, und die Nutzung von Datenaufnahme, Datenabruf und verlängerter Aufbewahrung für alle On-Demand-Streams ist mindestens 60% niedriger als bei On-Demand-Standard.
- Bereitgestellt — Für Datenstreams mit einem Bereitstellungsmodus müssen Sie die Anzahl der Shards für den Datenstrom angeben. Die Gesamtkapazität eines Datenstroms ist die Summe der Kapazitäten der einzelnen Shards. Sie können die Anzahl der Shards in einem Datenstream nach Bedarf erhöhen oder verringern.

Sie können Kinesis Data Streams verwenden `PutRecord` und `PutRecords` APIs in jedem Modus Daten in Ihre Datenströme schreiben. Zum Abrufen von Daten unterstützen alle drei Modi Standardverbraucher, die die `GetRecords` API verwenden, und Enhanced Fan-Out (EFO) - Verbraucher, die die API verwenden. `SubscribeToShard`

Alle Funktionen von Kinesis Data Streams, einschließlich Aufbewahrungsmodus, Verschlüsselung, Überwachungsmetriken und andere, werden sowohl für den On-Demand-Modus als auch für den Bereitstellungsmodus unterstützt. Kinesis Data Streams bietet eine hohe Beständigkeit und Verfügbarkeit sowohl im On-Demand-Modus als auch im Bereitstellungsmodus.

Funktionen und Anwendungsfälle im On-Demand-Standardmodus

Datenströme im On-Demand-Modus erfordern keine Kapazitätsplanung und werden automatisch skaliert, sodass sie einen Schreib- und Lesedurchsatz von mehreren Gigabyte pro Minute verarbeiten können. Der On-Demand-Modus vereinfacht die Aufnahme und Speicherung großer Datenmengen bei niedriger Latenz, da er die Bereitstellung und Verwaltung von Servern, Speicher oder Durchsatz überflüssig macht. Sie können Milliarden von Datensätzen pro Tag aufnehmen, ohne dass der Betrieb beeinträchtigt wird.

Der On-Demand-Modus ist ideal, um den Anforderungen eines stark variablen und unvorhersehbaren Anwendungsverkehrs gerecht zu werden. Sie müssen diese Workloads nicht mehr für Spitzenkapazitäten bereitstellen, was bei geringer Auslastung zu höheren Kosten führen kann. Der On-Demand-Modus eignet sich für Workloads mit unvorhersehbaren und stark variablen Datenverkehrsmustern.

Im On-Demand-Kapazitätsmodus zahlen Sie pro GB geschriebener und gelesener Daten aus Ihren Datenströmen. Sie müssen nicht angeben, wie viel Lese- und Schreibdurchsatz Sie von Ihrer Anwendung erwarten. Kinesis Data Streams passt sich sofort an Ihre Workloads an, wenn diese ansteigen oder sinken. Weitere Informationen finden Sie unter Amazon Kinesis Data Streams – Preise.

Ein Datenstrom im On-Demand-Modus ermöglicht bis zu doppelt so viel wie den Spitzenschreibdurchsatz, der in den letzten 30 Tagen beobachtet wurde. Wenn der Schreibdurchsatz Ihres Datenstroms einen neuen Höchstwert erreicht, skaliert Kinesis Data Streams die Kapazität des Datenstroms automatisch. Zum Beispiel, wenn Ihr Datenstrom einen Schreibdurchsatz hat, der zwischen 10 MB/s und 40% MB/s, then Kinesis Data Streams ensures that you can easily burst to double your previous peak throughput, or 80 MB/s. If the same data stream sustains a new peak throughput of 50 MB/s, Kinesis Data Streams ensures that there is enough capacity to ingest 100 MB/s des Schreibdurchsatzes variiert. Eine Drosselung des Schreibvorgangs kann jedoch

auftreten, wenn Ihr Datenverkehr innerhalb von 15 Minuten auf mehr als das Doppelte des vorherigen Spitzenwerts ansteigt. Sie müssen diese gedrosselten Anfragen erneut versuchen.

Die aggregierte Lesekapazität eines Datenstroms im On-Demand-Modus steigt proportional zum Schreibdurchsatz. Auf diese Weise wird sichergestellt, dass Verbraucheranwendungen immer über einen ausreichenden Lesedurchsatz verfügen, um eingehende Daten in Echtzeit zu verarbeiten. Der Schreibdurchsatz ist mindestens doppelt so hoch wie beim Lesen von Daten mit der `GetRecords`-API. Wir empfehlen, dass Sie eine Verbraucheranwendung mit der `GetRecord`-API verwenden, damit sie genügend Spielraum hat, um aufzuholen, wenn die Anwendung sich von einer Ausfallzeit erholen muss. Es wird empfohlen, die Funktion Erweitertes Rundsenden von Kinesis Data Streams für Szenarien zu verwenden, in denen mehr als eine Verbraucheranwendung hinzugefügt werden muss. Erweitertes Rundsenden unterstützt das Hinzufügen von bis zu 20 Verbraucheranwendungen zu einem Datenstream mithilfe der `SubscribeToShard`-API, wobei jede Verbraucheranwendung über einen eigenen Durchsatz verfügt.

Behandeln Sie Ausnahmen beim Lese- und Schreibdurchsatz

Im On-Demand-Modus (genau wie im Modus für bereitgestellte Kapazität) müssen Sie für jeden Datensatz einen Partitionsschlüssel angeben, um Daten in Ihren Datenstream zu schreiben. Kinesis Data Streams verwendet Ihre Partitionsschlüssel, um Daten auf Shards zu verteilen. Kinesis Data Streams überwacht den Verkehr für jeden Shard. Wenn der eingehende Datenverkehr 500 KB/s pro Shard überschreitet, wird der Shard innerhalb von 15 Minuten aufgeteilt. Die Hash-Schlüsselwerte des übergeordneten Shards werden gleichmäßig auf die untergeordneten Shards verteilt.

Wenn der eingehende Datenverkehr das Doppelte des vorherigen Spitzenwertes übersteigt, kann es für etwa 15 Minuten zu Lese- oder Schreibausfällen kommen, selbst wenn die Daten gleichmäßig auf die Shards verteilt sind. Wir empfehlen, dass Sie alle derartigen Anfragen erneut versuchen, damit alle Datensätze ordnungsgemäß in Kinesis Data Streams gespeichert werden.

Es kann zu Lese- und Schreibausnahmen kommen, wenn Sie einen Partitionsschlüssel verwenden, der zu einer ungleichmäßigen Datenverteilung führt, und die einem bestimmten Shard zugewiesenen Datensätze dessen Grenzwerte überschreiten. Im On-Demand-Modus passt sich der Datenstrom automatisch an ungleichmäßige Datenverteilungsmuster an, es sei denn, ein einzelner Partitionsschlüssel überschreitet die Grenzwerte für einen MB/s Durchsatz von 1 Shard und 1000 Datensätze pro Sekunde.

Im On-Demand-Modus teilt Kinesis Data Streams die Shards gleichmäßig auf, wenn ein Anstieg des Datenverkehrs festgestellt wird. Hash-Schlüssel, die einen höheren Anteil des eingehenden Datenverkehrs an einen bestimmten Shard weiterleiten, werden jedoch nicht erkannt und isoliert.

Wenn Sie stark ungleichmäßige Partitionsschlüssel verwenden, erhalten Sie möglicherweise weiterhin Schreibausnahmen. Für solche Anwendungsfälle empfehlen wir, den Modus für bereitgestellte Kapazität zu verwenden, der granulare Shard-Splits unterstützt.

Funktionen und Anwendungsfälle des Advantage-Modus auf Abruf

On-Demand-Vorteil ist eine Einstellung auf Kontoebene, die mehr Funktionen freischaltet und eine andere Preisstruktur für alle On-Demand-Streams in der Region bietet. In diesem Modus behalten On-Demand-Streams ihre Funktionalität bei und skalieren die Kapazität weiterhin automatisch auf der Grundlage der tatsächlichen Datennutzung. Wenn Sie die Schreibdurchsatzkapazität eines Streams proaktiv erhöhen möchten, können Sie den Warmdurchsatz konfigurieren. Wenn Ihr Datenstream beispielsweise einen Schreibdurchsatz zwischen 10 MB/s und 40 hat, MB/s, you can expect it to handle up to 80MB/s of instant throughput increases without throttling. However, if you forecast an upcoming event to peak around 200MB/s of traffic, you can configure the stream with a warm throughput of 200MB/s um sicherzustellen, dass die Kapazität verfügbar ist, wenn der Datendurchsatz eintrifft. Bei Verwendung eines warmen Durchsatzes fallen keine zusätzlichen Kosten an.

Ein weiterer Vorteil des On-Demand-Advantage-Modus besteht darin, dass On-Demand-Streams zu einer einfacheren Preisstruktur übergehen. Wenn der Modus aktiviert ist, fallen für das Konto keine festen Gebühren pro Stream mehr an, und Sie müssen sich nur noch um die Datenaufnahme, den Datenabruf und die optionalen Gebühren für die erweiterte Aufbewahrung kümmern. Für jede Preisdimension gibt es zudem einen erheblichen Preisnachlass im Vergleich zur entsprechenden Dimension in On-Demand-Standard. Weitere Informationen finden Sie unter Amazon Kinesis Data Streams – Preise.

Für erweiterte Fan-Out-Datenabrufe fällt im Vergleich zu den Standarddatenabrufen in diesem Modus auch kein Aufpreis an. Darüber hinaus können Sie im On-Demand-Advantage-Modus bis zu 50 Verbraucher pro Stream registrieren, um das erweiterte Fan-Out zu nutzen. Durch die Aktivierung von On-Demand Advantage ist das Konto verpflichtet, mindestens 25% MiB/s of data ingest and 25MiB/s der Daten aus allen On-Demand-Streams abzurufen. Für Konten, die die Mindestnutzungsanforderungen erfüllen, überprüft die Kinesis Data Streams Streams-Konsole, ob die Nutzungsmuster Ihres Kontos für den On-Demand-Advantage-Modus geeignet sind.

Wenn die Datennutzung Ihres Kontos unter den Anforderungen liegt, wird Ihnen die Differenz als Fehlbetrag in Rechnung gestellt, es wird jedoch immer noch derselbe ermäßigte Tarif berechnet. Wenn Sie On-Demand Advantage aktivieren, müssen Sie außerdem mindestens 24 Stunden warten, bis Sie den Modus deaktivieren können. Insgesamt ist On-Demand Advantage die beste Methode,

um mit Kinesis Data Streams zu streamen, wenn Sie eine konsistente Durchsatznutzung haben, die der Mindestverpflichtung nahe oder darüber liegt, viele Fan-Out-Nutzer benötigen oder mit Hunderten von Datenströmen arbeiten.

Funktionen und Anwendungsfälle im Bereitstellungsmodus

Im Bereitstellungsmodus können Sie, nachdem Sie den Datenstream erstellt haben, Ihre Shard-Kapazität mithilfe der oder der AWS-Managementkonsole API dynamisch nach oben oder unten skalieren. UpdateShardCount Sie können Aktualisierungen vornehmen, während eine Produzenten- oder Konsumentenanwendung von Kinesis Data Streams Daten in den Stream schreibt oder aus dem Stream liest.

Der Bereitstellungsmodus eignet sich für vorhersehbaren Datenverkehr mit leicht zu prognostizierenden Kapazitätsanforderungen. Sie können den Bereitstellungsmodus verwenden, wenn Sie eine genaue Kontrolle darüber haben möchten, wie Daten auf Shards verteilt werden.

Mit einem Bereitstellungsmodus müssen Sie die Anzahl der Shards für den Datenstrom angeben. Zum Festlegen der Größe eines Datenstroms mit dem Bereitstellungsmodus benötigen Sie die folgenden Eingabewerte:

- Die durchschnittliche Größe des Datensatzes, der in den Stream geschrieben wird, in Kilobyte (KB), gerundet auf das nächste ganze Kilobyte (`average_data_size_in_KB`).
- Die Anzahl der Datensätze, die pro Sekunde in den Stream geschrieben bzw. daraus gelesen werden (`records_per_second`).
- Die Anzahl der Verbraucher, die Anwendungen von Kinesis Data Streams sind, die gleichzeitig und unabhängig voneinander Daten aus dem Stream konsumieren (`number_of_consumers`).
- Die Bandbreite für eingehende Schreiboperationen in KB (`incoming_write_bandwidth_in_KB`). Diese ist gleich `average_data_size_in_KB` multipliziert mit `records_per_second`.
- Die Bandbreite für ausgehende Leseoperationen in KB (`outgoing_read_bandwidth_in_KB`). Diese ist gleich `incoming_write_bandwidth_in_KB` multipliziert mit `number_of_consumers`.

Sie können die Anzahl der Shards (`number_of_shards`), die der Stream benötigt, mit den Eingabewerte in der folgenden Formel berechnen.

```
number_of_shards = ceiling(max(incoming_write_bandwidth_in_KiB/1024,
 outgoing_read_bandwidth_in_KiB/2048))
```

Im Bereitstellungsmodus kann es immer noch zu Ausnahmen beim Lese- und Schreibdurchsatz kommen, wenn Sie Ihren Datenstrom nicht für den Spitzendurchsatz konfigurieren. In diesem Fall müssen Sie Ihren Datenstrom manuell skalieren, um den Datenverkehr zu bewältigen.

Es kann auch zu Lese- und Schreibausnahmen kommen, wenn Sie einen Partitionsschlüssel verwenden, der zu einer ungleichmäßigen Datenverteilung führt, und die einem Shard zugewiesenen Datensätze dessen Grenzwerte überschreiten. Um dieses Problem im Bereitstellungsmodus zu beheben, identifizieren Sie solche Shards und teilen Sie sie manuell auf, um Ihren Datenverkehr besser zu bewältigen. Weitere Informationen finden Sie unter Resharding eines Streams.

Wechseln Sie zwischen den Modi

Für jeden Datenstrom in Ihrem AWS-Konto können Sie innerhalb von 24 Stunden zweimal zwischen dem On-Demand-Modus und dem Bereitstellungsmodus wechseln. Der Wechsel zwischen den Modi verursacht keine Unterbrechungen Ihrer Anwendungen, die diesen Datenstrom verwenden. Sie können weiterhin in diesen Datenstrom schreiben und aus ihm lesen. Wenn Sie zwischen den Modi wechseln, entweder von On-Demand-Modus zu Bereitgestellt oder von Bereitgestellt zu On-Demand-Modus, wird der Status des Streams auf Aktualisierung gesetzt. Sie müssen warten, bis der Status des Datenstroms den Status Aktiv erreicht hat, bevor Sie seine Eigenschaften erneut ändern können.

Wenn Sie vom Bereitstellungs- zum On-Demand-Kapazitätsmodus wechseln, behält Ihr Datenstrom zunächst die Shard-Anzahl bei, die er vor der Umstellung hatte. Ab diesem Zeitpunkt überwacht Kinesis Data Streams Ihren Datenverkehr und skaliert die Anzahl der Shards dieses On-Demand-Datenstroms in Abhängigkeit von Ihrem Schreibdurchsatz. Wenn Sie vom On-Demand-Modus in den Bereitstellungsmodus wechseln, behält Ihr Datenstream zunächst auch die Shard-Anzahl bei, die er vor der Umstellung hatte. Ab diesem Zeitpunkt sind Sie jedoch dafür verantwortlich, die Shard-Anzahl dieses Datenstroms zu überwachen und anzupassen, um Ihrem Schreibdurchsatz gerecht zu werden.

Sie können vom On-Demand-Standardmodus zum On-Demand-Advantage-Modus wechseln, indem Sie eine Einstellung auf Kontoebene aktivieren. Wenn diese Option aktiviert ist, verpflichtet sich das Konto, mindestens 25% MiB/s of data ingest and 25MiB/s des Datenabrufs in allen On-Demand-Streams in der Region zu nutzen. Nach der Aktivierung müssen Sie mindestens 24 Stunden warten, bevor Sie On-Demand Advantage deaktivieren können. Sie können die Änderung jedoch jederzeit beantragen. Wenn Sie von On-Demand-Advantage zu On-Demand-Standard wechseln möchten, müssen Sie zunächst alle für Demand-Streams konfigurierten warmen Durchsatz entfernen.

Erstellen Sie einen Stream mit dem AWS-Managementkonsole

Sie können einen Stream mithilfe der Konsole von Kinesis Data Streams, der API von Kinesis Data Streams oder der AWS Command Line Interface (AWS CLI) erstellen.

So erstellen Sie einen Datenstrom mit der Konsole

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Kinesis-Konsole unter https://console.aws.amazon.com/kinesis.
2. Erweitern Sie in der Navigationsleiste die Regionsauswahl und wählen Sie eine Region aus.
3. Klicken Sie auf Create data stream (Daten-Stream erstellen).
4. Geben Sie auf der Seite Kinesis-Stream erstellen einen Namen für Ihren Datenstrom ein und wählen Sie dann entweder den Kapazitätsmodus On-Demand oder Bereitgestellt. Der Modus On-Demand ist standardmäßig ausgewählt. Weitere Informationen finden Sie unter Wählen Sie den richtigen Modus zum Streamen.

   Im Modus On-Demand können Sie dann Kinesis-Stream erstellen wählen, um Ihren Datenstrom zu erstellen. Im Modus Bereitgestellt müssen Sie dann die Anzahl der benötigten Shards angeben und dann Kinesis-Stream erstellen auswählen.

   Auf der Seite Kinesis streams (Kinesis-Streams) wird für den Wert Status des Streams Creating (Erstellen) angezeigt, während der Stream erstellt wird. Sobald der Stream verwendet werden kann, ändert sich der Wert von Status in Active (Aktiv).
5. Wählen Sie den Namen des Streams aus. Auf der Seite Stream Details (Stream-Details) wird eine Zusammenfassung der Stream-Konfiguration zusammen mit Überwachungsinformationen angezeigt.

So erstellen Sie einen Stream mit der Kinesis-Data-Streams-API

- Weitere Informationen zum Erstellen eines Streams mit der API von Kinesis Data Streams finden Sie unter Erstellen Sie einen Stream mit dem APIs.

Um einen Stream mit dem zu erstellen AWS CLI

- Informationen zum Erstellen eines Streams mit dem AWS CLI finden Sie im Befehl create-stream.

Erstellen Sie einen Stream mit dem APIs

Führen Sie die folgenden Schritte aus, um Ihren Kinesis-Datenstrom zu erstellen.

Den Kinesis Data Streams Streams-Client erstellen

Bevor Sie mit Kinesis-Datenströmen arbeiten können, müssen Sie ein Client-Objekt erstellen. Der folgende Java-Code instanziiert einen Client-Builder und verwendet ihn zum Festlegen der Region, der Anmeldeinformationen und der Client-Konfiguration. Anschließend erstellt er ein Client-Objekt.

```
AmazonKinesisClientBuilder clientBuilder = AmazonKinesisClientBuilder.standard();

clientBuilder.setRegion(regionName);
clientBuilder.setCredentials(credentialsProvider);
clientBuilder.setClientConfiguration(config);

AmazonKinesis client = clientBuilder.build();
```

Weitere Informationen finden Sie unter Regionen und Endpunkte der Kinesis Data Streams im Allgemeine AWS-Referenz.

Erstellen Sie den Stream

Nachdem Sie Ihren Kinesis Data Streams Streams-Client erstellt haben, können Sie einen Stream mithilfe der Konsole oder programmgesteuert erstellen. Um einen Stream programmgesteuert zu erstellen, instanziieren Sie ein `CreateStreamRequest` Objekt und geben Sie einen Namen für den Stream an. Wenn Sie den Bereitstellungsmodus verwenden möchten, geben Sie die Anzahl der Shards an, die der Stream verwenden soll.

- On-Demand-Modus:

```
CreateStreamRequest createStreamRequest = new CreateStreamRequest();
createStreamRequest.setStreamName( myStreamName );
```

- Bereitgestellt:

```
CreateStreamRequest createStreamRequest = new CreateStreamRequest();
createStreamRequest.setStreamName( myStreamName );
createStreamRequest.setShardCount( myStreamSize );
```

Der Streamname bezeichnet den Stream. Der Name ist auf das AWS Konto beschränkt, das von der Anwendung verwendet wird. Er ist auch nach Region beschränkt. Das heißt, zwei Streams in zwei verschiedenen AWS Konten können denselben Namen haben, und zwei Streams in demselben AWS Konto, aber in zwei verschiedenen Regionen können denselben Namen haben, aber nicht zwei Streams auf demselben Konto und in derselben Region.

Der Durchsatz des Streams hängt von der Anzahl der Shards ab. Für einen höheren bereitgestellten Durchsatz benötigen Sie mehr Shards. Mehr Shards erhöhen auch die Kosten, die für den Stream AWS anfallen. Weitere Informationen zum Berechnen einer ausreichenden Anzahl von Shards für Ihre Anwendung finden Sie unter Wählen Sie den richtigen Modus zum Streamen.

Nachdem Sie das `createStreamRequest` Objekt konfiguriert haben, erstellen Sie einen Stream, indem Sie die `createStream` Methode auf dem Client aufrufen. Warten Sie nach dem Aufrufen von `createStream`, bis der Stream den Status `ACTIVE` erreicht hat, bevor Sie eine Operation auf dem Stream ausführen. Rufen Sie die `describeStream`-Methode auf, um den Status des Streams zu überprüfen. Allerdings löst `describeStream` eine Ausnahme aus, wenn der Stream nicht existiert. Aus diesem Grund müssen Sie den Aufruf `describeStream` in einen `try/catch`-Block einschließen.

```
client.createStream( createStreamRequest );
DescribeStreamRequest describeStreamRequest = new DescribeStreamRequest();
describeStreamRequest.setStreamName( myStreamName );

long startTime = System.currentTimeMillis();
long endTime = startTime + ( 10 * 60 * 1000 );
while ( System.currentTimeMillis() < endTime ) {
  try {
    Thread.sleep(20 * 1000);
  }
  catch ( Exception e ) {}

  try {
    DescribeStreamResult describeStreamResponse =
 client.describeStream( describeStreamRequest );
    String streamStatus =
 describeStreamResponse.getStreamDescription().getStreamStatus();
    if ( streamStatus.equals( "ACTIVE" ) ) {
      break;
    }
    //
    // sleep for one second
```

```
    //
    try {
      Thread.sleep( 1000 );
    }
    catch ( Exception e ) {}
  }
  catch ( ResourceNotFoundException e ) {}
}
if ( System.currentTimeMillis() >= endTime ) {
  throw new RuntimeException( "Stream " + myStreamName + " never went active" );
}
```

Aktualisieren Sie einen Stream

Sie können die Details eines Streams mithilfe der Konsole von Kinesis Data Streams, der API von Kinesis Data Streams oder der AWS CLI aktualisieren.

> **Note**
>
> Sie können die serverseitige Verschlüsselung für vorhandene Streams oder für vor kurzem erstellte Streams aktivieren.

Verwenden der Konsole

So aktualisieren Sie einen Daten-Stream mit der Konsole

1. Öffnen Sie die Amazon Kinesis Kinesis-Konsole unter https://console.aws.amazon.com/kinesis/.
2. Erweitern Sie in der Navigationsleiste die Regionsauswahl und wählen Sie eine Region aus.
3. Wählen Sie den Namen des Streams in der Liste. Auf der Seite Stream Details (Stream-Details) werden eine Zusammenfassung der Stream-Konfiguration und Überwachungsinformationen angezeigt.
4. Wählen Sie auf der Registerkarte Konfiguration die Option Kapazitätsmodus bearbeiten aus, um für einen Datenstrom zwischen den Modi On-Demand-Kapazität und Bereitgestellte Kapazität zu wechseln. Weitere Informationen finden Sie unter Wählen Sie den richtigen Modus zum Streamen.

> **Important**
>
> Für jeden Datenstream in Ihrem AWS Konto können Sie innerhalb von 24 Stunden zweimal zwischen dem On-Demand-Modus und dem Bereitstellungsmodus wechseln.

5. Um für einen Datenstrom im Bereitstellungsmodus die Anzahl der Shards zu bearbeiten, wählen Sie auf der Registerkarte Konfiguration die Option Bereitgestellte Shards bearbeiten aus und geben Sie dann eine neue Shard-Anzahl ein.
6. Wählen Sie zum Aktivieren der serverseitigen Verschlüsselung von Datensätzen die Option Edit (Bearbeiten) im Abschnitt Server-side encryption (Serverseitige Verschlüsselung) aus. Wählen Sie einen KMS-Schlüssel aus, der als Masterschlüssel für die Verschlüsselung verwendet werden soll, oder nutzen Sie den Standard-Masterschlüssel aws/kinesis, der von Kinesis verwaltet wird. Wenn Sie die Verschlüsselung für einen Stream aktivieren und Ihren eigenen AWS KMS Masterschlüssel verwenden, stellen Sie sicher, dass Ihre Producer- und Consumer-Anwendungen Zugriff auf den von Ihnen verwendeten AWS KMS Masterschlüssel haben. Informationen zum Zuweisen von Berechtigungen zu Anwendungen für den Zugriff auf einen benutzergenerierten AWS KMS -Schlüssel finden Sie unter the section called “Berechtigungen zur Verwendung von benutzergenerierten KMS-Schlüsseln”.
7. Zum Bearbeiten des Datenaufbewahrungszeitraums wählen Sie die Option Edit (Bearbeiten) im Abschnitt Data retention period (Datenaufbewahrungszeitraum) aus und geben dann den neuen Zeitraum ein.
8. Wenn Sie benutzerdefinierte Metriken in Ihrem Konto aktiviert haben, wählen Sie die Option Edit (Bearbeiten) im Abschnitt Shard level metrics (Metriken auf Shard-Ebene) aus und geben anschließend Metriken für Ihren Stream an. Weitere Informationen finden Sie unter the section called “Überwachen Sie den Kinesis Data Streams Streams-Dienst mit CloudWatch”.

Verwenden der API

Informationen zum Aktualisieren von Stream-Details mit der API finden Sie unter den folgenden Methoden:

- AddTagsToStream
- DecreaseStreamRetentionPeriod
- DisableEnhancedMonitoring
- EnableEnhancedMonitoring

- IncreaseStreamRetentionPeriod
- RemoveTagsFromStream
- StartStreamEncryption
- StopStreamEncryption
- UpdateShardCount

Verwenden Sie den AWS CLI

Informationen zum Aktualisieren eines Streams mit dem AWS CLI finden Sie in der Kinesis CLI-Referenz.

Auflisten von Streams

Streams sind auf das AWS Konto beschränkt, das mit den AWS Anmeldeinformationen verknüpft ist, die zur Instanziierung des Kinesis Data Streams Streams-Clients verwendet wurden, sowie auf die für den Client angegebene Region. In einem AWS -Konto könnten viele Streams gleichzeitig aktiv sein. Sie können Ihre Streams in der Konsole von Kinesis Data Streams oder programmgesteuert auflisten. Der Code in diesem Abschnitt zeigt, wie Sie alle Streams für Ihr Konto auflisten. AWS

```
ListStreamsRequest listStreamsRequest = new ListStreamsRequest();
listStreamsRequest.setLimit(20);
ListStreamsResult listStreamsResult = client.listStreams(listStreamsRequest);
List<String> streamNames = listStreamsResult.getStreamNames();
```

In diesem Codebeispiel wird zuerst eine neue Instance von `ListStreamsRequest` erstellt und die zugehörige `setLimit`-Methode aufgerufen, um festzulegen, dass maximal 20-Streams für die einzelnen Aufrufe von `listStreams` zurückgegeben werden sollen. Wenn Sie keinen Wert für `setLimit` festlegen, gibt Kinesis Data Streams eine Anzahl von Streams kleiner oder gleich der Anzahl im Konto zurück. Der Code übergibt `listStreamsRequest` anschließend an die `listStreams`-Methode des Clients. Der Rückgabewert `listStreams` wird in einem `ListStreamsResult`-Objekt gespeichert. Der Code ruft die `getStreamNames`-Methode auf diesem Objekt auf und speichert den zurückgegebenen Stream-Namen in der Liste `streamNames`. Beachten Sie, dass Kinesis Data Streams möglicherweise weniger Streams zurückgibt als durch das angegebene Limit festgelegt, selbst wenn es mehr Streams als die im Konto und in der Region gibt. Um sicherzustellen, dass alle Streams abgerufen werden, verwenden Sie die `getHasMoreStreams`-Methode, wie im nächsten Codebeispiel beschrieben.

```
while (listStreamsResult.getHasMoreStreams())
{
    if (streamNames.size() > 0) {
      listStreamsRequest.setExclusiveStartStreamName(streamNames.get(streamNames.size()
 - 1));
    }
    listStreamsResult = client.listStreams(listStreamsRequest);
    streamNames.addAll(listStreamsResult.getStreamNames());
}
```

Dieser Code ruft die `getHasMoreStreams`-Methode für `listStreamsRequest` auf, um zu überprüfen, ob zusätzlich zu den im ersten Aufruf von `listStreams` zurückgegebenen Streams weitere verfügbar sind. Wenn dies der Fall ist, ruft der Code die `setExclusiveStartStreamName`-Methode mit dem Namen des letzten Streams auf, der im vorherigen Aufruf von `listStreams` zurückgegeben wurde. Die `setExclusiveStartStreamName`-Methode bewirkt, dass der nächste Aufruf von `listStreams` nach diesem Stream beginnt. Die Gruppe der von diesem Aufruf zurückgegebenen Stream-Namen wird dann zur Liste `streamNames` hinzugefügt. Dieser Vorgang wird so lange fortgesetzt, bis alle Stream-Namen in der Liste erfasst wurden.

Die von `listStreams` zurückgegebenen Streams können einen der folgenden Status aufweisen:

- `CREATING`
- `ACTIVE`
- `UPDATING`
- `DELETING`

Sie können den Status eines Streams mit der `describeStream`-Methode überprüfen, wie im vorherigen Abschnitt Erstellen Sie einen Stream mit dem APIs dargestellt.

Shards auflisten

Ein Datenstrom kann einen oder mehrere Shards aufweisen. Die empfohlene Methode zum Auflisten oder Abrufen der Shards aus einem Datenstream ist die Verwendung der API. ListShards Das folgende Beispiel zeigt, wie Sie eine Liste der Shards in einem Datenstrom erhalten. Eine vollständige Beschreibung der in diesem Beispiel verwendeten Hauptoperation und aller Parameter, die Sie für die Operation festlegen können, finden Sie unter. ListShards

```
import software.amazon.awssdk.services.kinesis.KinesisAsyncClient;
import software.amazon.awssdk.services.kinesis.model.ListShardsRequest;
import software.amazon.awssdk.services.kinesis.model.ListShardsResponse;

import java.util.concurrent.TimeUnit;

public class ShardSample {

    public static void main(String[] args) {

        KinesisAsyncClient client = KinesisAsyncClient.builder().build();

        ListShardsRequest request = ListShardsRequest
                .builder().streamName("myFirstStream")
                .build();

        try {
            ListShardsResponse response = client.listShards(request).get(5000,
 TimeUnit.MILLISECONDS);
            System.out.println(response.toString());
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Zum Ausführen des vorherigen Codebeispiels können Sie eine POM-Datei wie die folgende verwenden.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/
xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>kinesis.data.streams.samples</groupId>
    <artifactId>shards</artifactId>
    <version>1.0-SNAPSHOT</version>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
```

```
                <artifactId>maven-compiler-plugin</artifactId>
                <configuration>
                    <source>8</source>
                    <target>8</target>
                </configuration>
            </plugin>
        </plugins>
    </build>
    <dependencies>
        <dependency>
            <groupId>software.amazon.awssdk</groupId>
            <artifactId>kinesis</artifactId>
            <version>2.0.0</version>
        </dependency>
    </dependencies>
</project>
```

Mit der `ListShards` API können Sie den ShardFilterParameter verwenden, um die Antwort der API herauszufiltern. Sie können jeweils nur einen Filter angeben.

Wenn Sie den `ShardFilter` Parameter beim Aufrufen der ListShards API verwenden, `Type` ist dies die erforderliche Eigenschaft und muss angegeben werden. Wenn Sie die Typen `AT_TRIM_HORIZON`, `FROM_TRIM_HORIZON`, oder `AT_LATEST` angeben, müssen Sie weder die optionalen Eigenschaften `ShardId` noch `Timestamp` angeben.

Wenn Sie den `AFTER_SHARD_ID`-Typ angeben, müssen Sie auch den Wert für die optionale `ShardId`-Eigenschaft angeben. Die `ShardId` Eigenschaft ist in ihrer Funktionalität identisch mit dem `ExclusiveStartShardId` Parameter der ListShards API. Wenn eine Eigenschaft `ShardId` angegeben ist, enthält die Antwort die Shards, beginnend mit dem Shard, dessen ID unmittelbar auf das von Ihnen angegebene `ShardId` folgt.

Wenn Sie den Typ `AT_TIMESTAMP` oder `FROM_TIMESTAMP_ID` angeben, müssen Sie auch den Wert für die optionale `Timestamp`-Eigenschaft angeben. Wenn Sie den Typ `AT_TIMESTAMP` angeben, werden alle Shards zurückgegeben, die zum angegebenen Zeitstempel geöffnet waren. Wenn Sie den Typ `FROM_TIMESTAMP` angeben, werden alle Shards ab dem angegebenen Zeitstempel bis TIP zurückgegeben.

> **Important**
>
> `DescribeStreamSummary`und `ListShard` APIs bieten eine skalierbarere Möglichkeit, Informationen über Ihre Datenströme abzurufen. Insbesondere die Kontingente für die

DescribeStream API können zu Drosselungen führen. Weitere Informationen finden Sie unter Kontingente und -Einschränkungen. Beachten Sie auch, dass `DescribeStream` Kontingente von allen Anwendungen gemeinsam genutzt werden, die mit allen Datenströmen in Ihrem AWS Konto interagieren. Die Kontingente für die ListShards API sind dagegen spezifisch für einen einzelnen Datenstrom. Mit der ListShards API erhalten Sie also nicht nur höhere TPS-Werte, sondern die Aktion lässt sich auch besser skalieren, je mehr Datenströme Sie erstellen.
Wir empfehlen Ihnen, all Ihre Produzenten und Verbraucher, die die DescribeStream API aufrufen, zu migrieren, sodass sie stattdessen die DescribeStreamSummary und die aufrufen. ListShard APIs Um diese Hersteller und Verbraucher zu identifizieren, empfehlen wir, Athena zum Analysieren von CloudTrail Protokollen zu verwenden, während Benutzeragenten für KPL und KCL in den API-Aufrufen erfasst werden.

```
SELECT useridentity.sessioncontext.sessionissuer.username,
useridentity.arn,eventname,useragent, count(*) FROM
cloudtrail_logs WHERE Eventname IN ('DescribeStream')  AND
eventtime
    BETWEEN ''
        AND ''
GROUP BY
 useridentity.sessioncontext.sessionissuer.username,useridentity.arn,eventname,useragent
ORDER BY  count(*) DESC LIMIT 100
```

Wir empfehlen außerdem, die AWS Lambda- und Amazon Firehose-Integrationen mit Kinesis Data Streams, die die `DescribeStream` API aufrufen, neu zu konfigurieren, sodass die Integrationen stattdessen und aufrufen. `DescribeStreamSummary` `ListShards` Insbesondere für AWS Lambda müssen Sie Ihre Ereignisquellenzuordnung aktualisieren. Für Amazon Firehose müssen die entsprechenden IAM-Berechtigungen aktualisiert werden, sodass sie die `ListShards`-IAM-Berechtigung enthalten.

Löschen Sie einen Stream

Sie können einen Stream mit der Konsole von Kinesis Data Streams oder programmgesteuert löschen. Um einen Stream programmgesteuert zu löschen, verwenden Sie `DeleteStreamRequest`, wie im folgenden Code gezeigt.

```
DeleteStreamRequest deleteStreamRequest = new DeleteStreamRequest();
deleteStreamRequest.setStreamName(myStreamName);
client.deleteStream(deleteStreamRequest);
```

Schließen Sie alle auf dem Stream ausgeführten Anwendungen, bevor Sie ihn löschen. Wenn eine Anwendung versucht, sich auf einem gelöschten Stream zu öffnen, empfängt sie `ResourceNotFound`-Ausnahmen. Wenn Sie darüber hinaus anschließend einen neuen Stream erstellen, der denselben Namen wie Ihr vorheriger Stream hat, und Anwendungen, die auf dem vorherigen Stream ausgeführt wurden, weiterhin ausgeführt werden, versuchen diese Anwendungen möglicherweise, mit dem neuen Stream zu interagieren, so als wäre es der vorherige Stream – was zu einem unerwarteten Verhalten führen könnte.

Einen Stream erneut teilen

> **Important**
>
> Sie können Ihren Stream mithilfe der API erneut teilen. UpdateShardCount Andernfalls können Sie auch weiterhin Teilungen und Zusammenführungen ausführen, wie hier beschrieben.

Amazon Kinesis Data Streams unterstützt das Resharding, das Ihnen ermöglicht, die Anzahl der Shards in Ihrem Stream zur Anpassung an Änderungen an der Durchflussrate der Daten im Stream anzupassen. Resharding ist eine erweiterte Operation. Wenn Sie noch keine Erfahrung mit Kinesis Data Streams haben, machen Sie sich zuerst mit allen anderen Aspekten von Kinesis Data Streams vertraut, bevor Sie zu diesem Thema zurückkehren.

Derzeit gibt es zwei Arten von Resharding-Vorgängen: Shard-Teilungen und Shard-Zusammenführungen. Bei einer Shard-Teilung wird ein einzelner Shard in zwei Shards geteilt. Bei einer Shard-Zusammenführung werden zwei Shards in einem einzelnen Shard zusammengeführt. Resharding erfolgt stets paarweise, d. h. dass in einem einzelnen Vorgang keine Teilungen in mehr als zwei Shards möglich sind und dass in einem einzelnen Vorgang keine Zusammenführungen von mehr als zwei Shards möglich sind. Der Shard oder das Shard-Paar, auf dem der Resharding-Vorgang ausgeführt wird, wird als übergeordneter Shard bezeichnet. Der Shard oder das Shard-Paar, der/das aus dem Resharding-Vorgang resultiert, wird als untergeordneter Shard bezeichnet.

Durch Teilungen erhöht sich die Anzahl der Shards in Ihrem Stream und damit die Datenkapazität des Streams. Da Gebühren pro Shard anfallen, erhöhen Teilungen die Kosten für Ihren Stream.

Gleichermaßen verringern Zusammenführungen die Anzahl der Shards in Ihrem Stream und damit die Datenkapazität – und Kosten – des Streams.

Das Resharding wird üblicherweise von einer administrativen Anwendung ausgeführt. Diese unterscheiden sich von den Produzentenanwendungen (zum Senden) und den Konsumentenanwendungen (zum Abrufen). Eine solche Verwaltungsanwendung überwacht die Gesamtleistung des Streams auf der Grundlage von Metriken, die von Amazon bereitgestellt werden, CloudWatch oder auf der Grundlage von Metriken, die von Herstellern und Verbrauchern gesammelt wurden. Die Verwaltungsanwendung benötigt außerdem ein breiteres Spektrum an IAM-Berechtigungen als die der Verbraucher oder Produzenten, da die Verbraucher und Produzenten normalerweise keinen Zugriff auf die für das Resharding APIs verwendeten Daten benötigen sollten. Weitere Informationen zu IAM-Berechtigungen für Kinesis Data Streams finden Sie unter Steuern des Zugriffs auf Amazon Kinesis Data Streams Streams-Ressourcen mithilfe von IAM.

Weitere Informationen zu Resharding finden Sie unter Wie ändere ich die Anzahl der offenen Shards in Kinesis Data Streams?

Themen

- Entscheiden Sie sich für eine Strategie für das Resharding
- Teilen Sie einen Shard auf
- Zwei Shards zusammenführen
- Schließen Sie die Resharding-Aktion ab

Entscheiden Sie sich für eine Strategie für das Resharding

Der Zweck des Reshardings in Amazon Kinesis Data Streams besteht darin, Ihrem Stream die Anpassung an Änderungen in der Durchflussrate von Daten zu ermöglichen. Sie teilen Shards, um die Kapazität (und Kosten) Ihres Streams zu erhöhen. Sie führen Shards zusammen, um die Kosten (und Kapazität) Ihres Streams zu verringern.

Ein Ansatz für das Resharding besteht darin, jeden Shard im Stream zu teilen – wodurch sich die Kapazität des Streams verdoppeln würde. Dadurch kann jedoch mehr zusätzliche Kapazität bereitgestellt werden, als Sie tatsächlich benötigen, was unnötige Kosten verursachen würde.

Sie können darüber hinaus Metriken verwenden, um zu ermitteln, welche Ihre heißen oder kalten Shards sind, d. h. Shards, die viel mehr bzw. viel weniger Daten empfangen als erwartet. Sie könnten dann selektiv die heißen Shards teilen, um die Kapazität für die Hash-Schlüssel zu erhöhen, die

auf solche Shards abzielen. Gleichermaßen könnten Sie kalte Shards zusammenführen, um deren ungenutzte Kapazitäten besser zu nutzen.

Sie können einige Leistungsdaten für Ihren Stream aus den CloudWatch Amazon-Metriken abrufen, die Kinesis Data Streams veröffentlicht. Sie können jedoch auch eigene Metriken für Ihre Streams erfassen. Ein Ansatz wäre die Protokollierung der Hash-Schlüsselwerte, die von den Partitionsschlüsseln für Ihre Datensätze generiert wurden. Beachten Sie, dass Sie den Partitionsschlüssel zu dem Zeitpunkt festlegen, an dem Sie den Datensatz zu dem Stream hinzufügen.

```
putRecordRequest.setPartitionKey( String.format( "myPartitionKey" ) );
```

Kinesis Data Streams verwendet MD5, um den Hash-Schlüssel aus dem Partitionsschlüssel zu berechnen. Da Sie den Partitionsschlüssel für den Datensatz angeben, könnten Sie ihn verwenden, MD5 um den Hash-Schlüsselwert für diesen Datensatz zu berechnen und zu protokollieren.

Sie könnten auch die IDs Shards protokollieren, denen Ihre Datensätze zugewiesen sind. Die Shard-ID ist verfügbar über die `getShardId`-Methode des `putRecordResults`-Objekts, das von der `putRecords`-Methode zurückgegeben wurde, und des `putRecordResult`-Objekts, das von der `putRecord`-Methode zurückgegeben wurde.

```
String shardId = putRecordResult.getShardId();
```

Anhand des Shards IDs und der Hash-Schlüsselwerte können Sie feststellen, welche Shards und Hash-Schlüssel den meisten oder den wenigsten Traffic erhalten. Anschließend können Sie mithilfe von Resharding mehr oder weniger Kapazitäten bereitstellen, wie für diese Schlüssel angemessen.

Teilen Sie einen Shard auf

Um einen Shard in Amazon Kinesis Data Streams zu teilen, müssen Sie festlegen, wie Hash-Schlüsselwerte aus dem übergeordneten Shard auf die untergeordneten Shards umverteilt werden sollten. Wenn Sie einen Datensatz zu einem Stream hinzufügen, wird dieser basierend auf einem Hash-Schlüsselwert einem Shard zugeordnet. Der Hash-Schlüsselwert ist der MD5Hash des Partitionsschlüssels, den Sie für den Datensatz angeben, wenn Sie den Datensatz zum Stream hinzufügen. Datensätze mit demselben Partitionsschlüssel haben auch denselben Hash-Schlüsselwert.

Die möglichen Hash-Schlüsselwerte für einen bestimmten Shard bilden eine Reihe von geordneten fortlaufenden nicht-negativen Ganzzahlen. Diese Reihe möglicher Hash-Schlüsselwerte ergibt sich wie folgt:

```
shard.getHashKeyRange().getStartingHashKey();
shard.getHashKeyRange().getEndingHashKey();
```

Wenn Sie die Shard teilen, legen Sie einen Wert in diesem Bereich fest. Dieser Hash-Schlüsselwert und alle höheren Hash-Schlüsselwerte werden auf eine der untergeordneten Shards verteilt. Alle niedrigeren Hash-Schlüsselwerte werden an die andere untergeordnete Shard verteilt.

Der folgende Code zeigt einen Shard -Teilungsvorgang, bei dem die Hash-Schlüssel gleichmäßig zwischen den untergeordneten Shards verteilt werden, wobei der übergeordnete Shard im Wesentlichen in zwei Hälften geteilt wird. Dies ist nur eine Möglichkeit, den übergeordneten Shard zu teilen. Sie könnten beispielsweise den Shard so teilen, dass das untere Drittel des Schlüssels aus dem übergeordneten Shard an einen untergeordneten Shard und die zwei oberen Drittel der Schlüssel an einen anderen untergeordneten Shard übergehen. Bei vielen Anwendungen ist jedoch das Teilen von Shards in zwei Hälften ein effektiver Ansatz.

Der Code setzt voraus, dass `myStreamName` den Namen Ihres Streams enthält und dass die Objektvariable `shard` den zu teilenden Shard enthält. Beginnen Sie mit der Instanziierung eines neuen `splitShardRequest`-Objekts und dem Festlegen des Stream-Namens und der Shard-ID.

```
SplitShardRequest splitShardRequest = new SplitShardRequest();
splitShardRequest.setStreamName(myStreamName);
splitShardRequest.setShardToSplit(shard.getShardId());
```

Ermitteln Sie den Hash-Schlüsselwert, der genau zwischen dem niedrigsten und dem höchsten Wert im Shard liegt. Dies ist der Hash-Startschlüsselwert für den untergeordneten Shard, der die obere Hälfte der Hash-Schlüssel aus dem übergeordneten Shard enthält. Geben Sie diesen Wert in der `setNewStartingHashKey`-Methode an. Sie müssen nur diesen Wert angeben. Kinesis Data Streams verteilt automatisch die Hash-Schlüssel unter diesem Wert an den anderen untergeordneten Shard, der bei dieser Teilung erstellt wird. Der letzte Schritt besteht darin, die `splitShard`-Methode auf dem Kinesis-Data-Streams-Client aufzurufen.

```
BigInteger startingHashKey = new
 BigInteger(shard.getHashKeyRange().getStartingHashKey());
BigInteger endingHashKey   = new
 BigInteger(shard.getHashKeyRange().getEndingHashKey());
```

```
String newStartingHashKey  = startingHashKey.add(endingHashKey).divide(new
 BigInteger("2")).toString();

splitShardRequest.setNewStartingHashKey(newStartingHashKey);
client.splitShard(splitShardRequest);
```

Der erste Schritt nach diesem Verfahren wird unter Warten Sie, bis ein Stream wieder aktiv wird veranschaulicht.

Zwei Shards zusammenführen

Bei einem Shard-Zusammenführungsvorgang werden zwei festgelegte Shards in einem einzelnen Shard kombiniert. Nach der Zusammenfassung empfängt der einzelne untergeordnete Shard Daten für alle Hash-Schlüsselwerte, die in den zwei übergeordneten Shards enthalten sind.

Shard-Nachbarschaft

Um zwei Shards zusammenführen zu können, müssen die Shards benachbart sein. Zwei Shards gelten als benachbart, wenn der Verband der Hash-Schlüsselbereiche für die beiden Shards eine fortlaufende Reihe ohne Lücken bildet. Wenn Sie zum Beispiel über zwei Shards verfügen, von denen einer einen Hash-Schlüsselbereich von 276 bis 381 und der andere einen Hash-Schlüsselbereich von 382 bis 454 aufweist, dann könnten Sie diese beiden Shards zu einem einzigen Shard zusammenführen, der über einen Hash-Schlüsselbereich von 276 bis 454 verfügen würde.

Wenn Sie in einem anderen Beispiel über zwei Shards verfügen, von denen einer einen Hash-Schlüsselbereich von 276 bis 381 und der andere einen Hash-Schlüsselbereich von 455 bis 560 aufweist, dann könnten Sie diese beiden Shards nicht zusammenführen, weil ein oder mehr weitere Shards zwischen diesen beiden liegen würden, die den Bereich von 382 bis 454 abdecken.

Der Satz aller `OPEN` Shards in einem Stream — als Gruppe — umfasst immer den gesamten Bereich der Hash-Schlüsselwerte. MD5 Weitere Informationen zu Shard-Status – wie beispielsweise `CLOSED` – finden Sie unter Denken Sie an Datenrouting, Datenpersistenz und den Shard-Status nach einem Reshard.

Zum Identifizieren von Shards, die zum Zusammenführen infrage kommen, sollten Sie alle Shards im Status `CLOSED` herausfiltern. Shards mit dem Status `OPEN` – also nicht `CLOSED` – weisen die Sequenzendzahl `null` auf. Sie können die Sequenzendzahl für einen Shard folgendermaßen testen:

```
if( null == shard.getSequenceNumberRange().getEndingSequenceNumber() )
{
```

```
  // Shard is OPEN, so it is a possible candidate to be merged.
}
```

Nachdem die geschlossenen Shards herausgefiltert wurden, sortieren Sie die verbleibenden Shards nach dem höchsten Hash-Schlüsselwert, der von den Shards jeweils unterstützt wird. Sie können diesen Wert folgendermaßen abrufen:

```
shard.getHashKeyRange().getEndingHashKey();
```

Wenn zwei Shards in dieser gefilterten, sortierten Liste benachbart sind, können sie zusammengeführt werden.

Code für den Zusammenführungsvorgang

Mit dem folgenden Code lassen sich zwei Shards zusammenführen. Der Code setzt voraus, dass `myStreamName` den Namen Ihres Streams enthält und dass die Objektvariablen `shard1` und `shard2` die zwei benachbarten zusammenzuführenden Shards enthalten.

Beginnen Sie für den Zusammenführungsvorgang mit der Instanziierung eines neuen `mergeShardsRequest`-Objekts. Geben Sie den Stream-Namen mit der `setStreamName`-Methode an. Geben Sie anschließend die zwei zusammenzuführenden Shards mithilfe der Methoden `setShardToMerge` und `setAdjacentShardToMerge` an. Rufen Sie anschließend die Methode `mergeShards` auf dem Client von Kinesis Data Streams aus, um den Vorgang auszuführen.

```
MergeShardsRequest mergeShardsRequest = new MergeShardsRequest();
mergeShardsRequest.setStreamName(myStreamName);
mergeShardsRequest.setShardToMerge(shard1.getShardId());
mergeShardsRequest.setAdjacentShardToMerge(shard2.getShardId());
client.mergeShards(mergeShardsRequest);
```

Der erste Schritt nach diesem Verfahren wird unter Warten Sie, bis ein Stream wieder aktiv wird veranschaulicht.

Schließen Sie die Resharding-Aktion ab

Nach allen Resharding-Vorgängen in Amazon Kinesis Data Streams und vor dem Fortsetzen der normalen Datensatzverarbeitung sind weitere Verfahren und Überlegungen erforderlich. In den folgenden Abschnitten werden diese beschrieben.

Themen

- Warten Sie, bis ein Stream wieder aktiv wird
- Denken Sie an Datenrouting, Datenpersistenz und den Shard-Status nach einem Reshard

Warten Sie, bis ein Stream wieder aktiv wird

Nachdem Sie einen Resharding-Vorgang (entweder `splitShard` oder) aufgerufen habenmergeShards, müssen Sie warten, bis der Stream wieder aktiv wird. Der zu verwendende Code entspricht dem Code beim Warten, bis ein Stream nach dem Erstellen eines Streams aktiviert wird. Dieser Code sieht wie folgt aus:

```
DescribeStreamRequest describeStreamRequest = new DescribeStreamRequest();
describeStreamRequest.setStreamName( myStreamName );

long startTime = System.currentTimeMillis();
long endTime = startTime + ( 10 * 60 * 1000 );
while ( System.currentTimeMillis() < endTime )
{
  try {
    Thread.sleep(20 * 1000);
  }
  catch ( Exception e ) {}

  try {
    DescribeStreamResult describeStreamResponse =
 client.describeStream( describeStreamRequest );
    String streamStatus =
 describeStreamResponse.getStreamDescription().getStreamStatus();
    if ( streamStatus.equals( "ACTIVE" ) ) {
      break;
    }
   //
    // sleep for one second
    //
    try {
      Thread.sleep( 1000 );
    }
    catch ( Exception e ) {}
  }
  catch ( ResourceNotFoundException e ) {}
}
if ( System.currentTimeMillis() >= endTime )
{
```

```
  throw new RuntimeException( "Stream " + myStreamName + " never went active" );
}
```

Denken Sie an Datenrouting, Datenpersistenz und den Shard-Status nach einem Reshard

Kinesis Data Streams ist ein Echtzeit-Datenstreaming-Dienst. Ihre Anwendungen sollten davon ausgehen, dass Daten kontinuierlich durch die Shards in Ihrem Stream fließen. Bei einem Resharding werden Datensätze, die an die übergeordneten Shards geleitet wurden, basierend auf den Hash-Schlüsselwerten, die den Partitionsschlüsseln zu den Datensätzen zugeordnet werden, zu den untergeordneten Shards umgeleitet. Datensätze, die sich vor dem Resharding in den übergeordneten Shards befunden haben, verbleiben jedoch in diesen Shards. Die übergeordneten Shards verschwinden nicht, wenn der Reshard auftritt. Sie bleiben zusammen mit den Daten erhalten, die sie vor dem Resharding enthielten. Die Datensätze in den übergeordneten Shards können mithilfe der Operationen `getShardIterator` und `getRecords` in der API für Kinesis Data Streams und über die Kinesis Client Library aufgerufen werden.

> **Note**
>
> Datensätze können ab dem Zeitpunkt, an dem Sie zum Stream hinzugefügt wurden, bis zum aktuellen Aufbewahrungszeitraum aufgerufen werden. Dies gilt unabhängig von Änderungen an den Shards in dem Stream in diesem Zeitraum. Weitere Informationen zum Aufbewahrungszeitraum eines Streams finden Sie unter Ändern Sie den Aufbewahrungszeitraum für Daten.

Bei einem Resharding-Vorgang geht ein übergeordneter Shard vom Status `OPEN` in den Status `CLOSED` und anschließend in den Status `EXPIRED` über.

- OPEN: Vor einem Resharding-Vorgang befindet sich ein übergeordneter Shard im Status `OPEN`. Dies bedeutet, dass Datensätze sowohl zum Shard hinzugefügt als auch vom Shard abgerufen werden können.
- CLOSED: Nach einem Reshard-Vorgang geht der übergeordnete Shard in den Status `CLOSED` über. Das bedeutet, dass Datensätze sind mehr zum Shard hinzugefügt werden. Die Datensätze, die zu diesem Shard hinzugefügt worden wären, werden nun stattdessen zu einem untergeordneten Shard hinzugefügt. Datensätze können jedoch weiterhin für einen begrenzten Zeitraum von dem Shard abgerufen werden.

- EXPIRED: Wenn der Aufbewahrungszeitraum des Streams abgelaufen ist, sind auch alle Datensätze im übergeordneten Shard abgelaufen und nicht mehr zugänglich. Zu diesem Zeitpunkt geht der Shard selbst in den Status `EXPIRED` über. Aufrufe an `getStreamDescription().getShards` zum Aufzählen der Shards im Stream berücksichtigen keine Shards mit dem Status `EXPIRED` in der Liste der zurückgegebenen Shards. Weitere Informationen zum Aufbewahrungszeitraum eines Streams finden Sie unter Ändern Sie den Aufbewahrungszeitraum für Daten.

Wenn sich der Stream nach einem Resharding-Vorgang wieder im Status `ACTIVE` befindet, können Sie sofort mit dem Auslesen von Daten aus den untergeordneten Shards beginnen. Die übergeordneten Shards, die nach dem Reshard verbleiben, können jedoch immer noch Daten enthalten, die Sie noch nicht gelesen haben und die dem Stream vor dem Reshard hinzugefügt wurden. Wenn Sie Daten aus den untergeordneten Shards auslesen, bevor alle Daten aus den übergeordneten Shards ausgelesen wurden, lesen Sie die Daten für einen bestimmten Hash-Schlüssel möglicherweise nicht in der durch die Sequenznummern der Datensätze festgelegten Reihenfolge. Unter der Annahme, dass die Reihenfolge der Daten wichtig ist, sollten Sie daher nach einem Resharding Daten aus den übergeordneten Shards immer so lange weiter auslesen, bis sie ausgeschöpft sind. Erst dann sollten Sie mit dem Auslesen von Daten aus den untergeordneten Shards beginnen. Wenn `getRecordsResult.getNextShardIterator` den Wert `null` zurückgibt, bedeutet dies, dass alle Daten im übergeordneten Shard ausgelesen wurden.

Ändern Sie den Aufbewahrungszeitraum für Daten

Amazon Kinesis Data Streams unterstützt Änderungen des Zeitraums der Datensatzaufbewahrung für einen Datenstrom. Ein Kinesis-Datenstrom ist eine sortierte Folge von Datensätzen, in die in Echtzeit geschrieben und aus denen in Echtzeit gelesen werden kann. Datensätze werden deshalb vorübergehend in Shards in Ihrem Stream gespeichert. Der Zeitraum ab dem Hinzufügen eines Datensatzes bis zu dem Zeitpunkt, an dem er nicht mehr verfügbar ist, wird als Aufbewahrungszeitraum bezeichnet. Ein Kinesis-Datenstrom speichert standardmäßig Datensätze von 24 Stunden bis zu 8 760 Stunden (365 Tage).

Sie können den Aufbewahrungszeitraum über die Kinesis Data Streams Streams-Konsole oder mithilfe der DecreaseStreamRetentionPeriodOperationen IncreaseStreamRetentionPeriodund den Vorgängen aktualisieren. Mit der Konsole von Kinesis Data Streams können Sie den Aufbewahrungszeitraum mehrerer Datenströme gleichzeitig bearbeiten. Sie können die Aufbewahrungsdauer mithilfe der IncreaseStreamRetentionPeriodOperation oder der Kinesis Data Streams Streams-Konsole auf maximal 8760 Stunden (365 Tage) erhöhen. Sie können die

Aufbewahrungsdauer mithilfe der DecreaseStreamRetentionPeriodOperation oder der Kinesis Data Streams Streams-Konsole auf mindestens 24 Stunden reduzieren. Die Anforderungssyntax für beide Operationen enthält den Stream-Namen und den Aufbewahrungszeitraum in Stunden. Zuletzt können Sie den aktuellen Aufbewahrungszeitraum eines Streams überprüfen, indem Sie die DescribeStream-Operation aufrufen.

Das folgende Beispiel zeigt die Änderung des Aufbewahrungszeitraums unter Verwendung der AWS CLI.

```
aws kinesis increase-stream-retention-period --stream-name retentionPeriodDemo --
retention-period-hours 72
```

Kinesis Data Streams hört innerhalb von einigen Minuten nach einer Erhöhung des Aufbewahrungszeitraums damit auf, Datensätze zum alten Aufbewahrungszeitraum unzugänglich zu machen. Beispielsweise bedeutet eine Änderung des Aufbewahrungszeitraums von 24 Stunden auf 48 Stunden, dass Datensätze, die 23 Stunden und 55 Minuten vorher zum Stream hinzugefügt wurden, nach 24 Stunden weiterhin verfügbar sind.

Kinesis Data Streams macht bei einer Verringerung des Aufbewahrungszeitraums beinahe sofort die Datensätze unzugänglich, die vor dem neuen Aufbewahrungszeitraum liegen. Deshalb sollten Sie beim Aufrufen der DecreaseStreamRetentionPeriod-Operation äußerst vorsichtig vorgehen.

Legen Ihren Zeitraum der Datenaufbewahrung so fest, dass im Fall von Problemen sichergestellt ist, dass Ihre Konsumenten Daten auslesen können, bevor diese auslaufen. Sie sollten alle Möglichkeiten sorgfältig prüfen, wie beispielsweise ein Problem mit Ihrer Datensatzverarbeitungslogik oder einer nachgelagerten Abhängigkeit, die für einen langen Zeitraum inaktiv ist. Der Aufbewahrungszeitraum kann als Sicherheitsnetz betrachtet werden, durch das Ihre Datenkonsumenten mehr Zeit für die Wiederherstellung haben. Die API-Operationen für Aufbewahrungszeiträume ermöglichen Ihnen, dies proaktiv einzurichten oder auf Betriebsereignisse zu reagieren.

Für Streams mit einem Aufbewahrungszeitraum von mehr als 24 Stunden fallen zusätzliche Gebühren an. Weitere Informationen finden Sie unter Preise für Amazon Kinesis Daten-Streams.

Kennzeichnen Sie Ihre Amazon Kinesis Data Streams Streams-Ressourcen

Sie können Streams und erweiterten Fan-Out-Consumern, die Sie in Amazon Kinesis Data Streams erstellen, Ihre eigenen Metadaten in Form von Tags zuweisen. Ein Tag ist ein Schlüssel-Wert-Paar, das Sie für einen Stream definieren. Die Verwendung von Tags ist eine einfache und dennoch leistungsstarke Möglichkeit, AWS Ressourcen zu verwalten und Daten, einschließlich Rechnungsdaten, zu organisieren.

Inhalt

- Lesen Sie die Grundlagen von Stichwörtern
- Verfolgen Sie die Kosten mithilfe von Tagging
- Verstehen Sie die Einschränkungen von Tags
- Taggen Sie Streams mit der Kinesis Data Streams Streams-Konsole
- Taggen Sie Streams mit dem AWS CLI
- Taggen von Streams mithilfe der Kinesis Data Streams APIs
- Taggen Sie Verbraucher mit dem AWS CLI
- Taggen Sie Verbraucher, die Kinesis Data Streams verwenden APIs

Lesen Sie die Grundlagen von Stichwörtern

Zu den Kinesis Data Streams Streams-Ressourcen, die Sie taggen können, gehören Datenstreams und erweiterte Fan-Out-Nutzer. Sie verwenden die Kinesis Data Streams Streams-Konsole oder die Kinesis Data Streams Streams-API AWS CLI, um die folgenden Aufgaben auszuführen:

- Erstellen Sie eine Ressource mit Tags
- Fügen Sie einer Ressource Tags hinzu
- Listet die Tags für Ihre Ressourcen auf
- Entfernen Sie Tags aus einer Ressource

> **Note**
>
> Mit der Kinesis Data Streams Streams-Konsole können Sie keine Tags auf erweiterte Fan-Out-Nutzer anwenden. Verwenden AWS CLI Sie unsere Kinesis Data Streams Streams-API, um Tags auf Verbraucher anzuwenden.

Sie können mit Tags auch Ihre -Ressourcen kategorisieren. Sie können Ressourcen beispielsweise nach Zweck, Inhaber oder Umgebung kategorisieren. Da Sie für jeden Tag den Schlüssel und Wert definieren, können Sie eine auf benutzerdefinierte Reihe von Kategorien anlegen, die Ihren jeweiligen Anforderungen gerecht wird. Sie können beispielsweise eine Reihe von Tags definieren, mit deren Hilfe Sie Ressourcen nach Eigentümer und zugehöriger Anwendung nachverfolgen können. Im Folgenden finden Sie einige Beispiele für Tags:

- Projekt: Projektname
- Inhaber: Name
- Zweck: Belastungstest
- Anwendung: Anwendungsname
- Umgebung: Produktion

> **Important**
>
> - Um beim Erstellen eines Streams Tags hinzuzufügen, müssen Sie die `kinesis:AddTagsToStream` Berechtigungen `kinesis:CreateStream` und für diesen Stream angeben. Sie können die **`kinesis:TagResource`** Berechtigung zum Markieren von Streams nicht verwenden, während Sie sie erstellen.
> - Um bei der Registrierung als Verbraucher Tags hinzuzufügen, müssen Sie die `kinesis:RegisterStreamConsumer` Berechtigungen `kinesis:TagResource` und angeben.

Verfolgen Sie die Kosten mithilfe von Tagging

Sie können Tags verwenden, um Ihre AWS Kosten zu kategorisieren und nachzuverfolgen. Wenn Sie Tags auf Ihre Kinesis Data Streams Streams-Ressourcen anwenden, enthält Ihr AWS

Kostenzuordnungsbericht die Nutzung und die Kosten, die nach Tags zusammengefasst sind. Sie können Tags verwenden, die Geschäftskategorien wie Kostenstellen, Anwendungsnamen oder Eigentümer repräsentieren, um Ihre Kosten für mehrere Services zu organisieren. Weitere Informationen finden Sie unter Verwenden von Kostenzuordnungs-Tags für benutzerdefinierte Fakturierungsberichte im AWS Billing -Benutzerhandbuch.

Verstehen Sie die Einschränkungen von Tags

Für Tags gelten die folgenden Einschränkungen:

Grundlegende Einschränkungen

- Die maximale Anzahl von Tags pro Ressource beträgt 50.
- Bei Tag-Schlüsseln und -Werten wird zwischen Groß- und Kleinschreibung unterschieden.
- Sie können Tags für eine gelöschte Ressource nicht ändern oder bearbeiten.

Einschränkungen für Tag-Schlüssel

- Jeder Tag-Schlüssel muss einmalig sein. Wenn Sie einen Tag mit einem Schlüssel hinzufügen, der bereits verwendet wird, wird das vorhandene Schlüssel-Wert-Paar durch den neuen Tag überschrieben.
- Sie können einen Tag-Schlüssel nicht mit beginnen`aws:`, da dieses Präfix für die Verwendung durch reserviert ist AWS. AWS erstellt in Ihrem Namen Tags, die mit diesem Präfix beginnen, aber Sie können sie nicht bearbeiten oder löschen.
- Tag-Schlüssel müssen zwischen 1 und 128 Unicode-Zeichen lang sein.
- Tag-Schlüssel müssen die folgenden Zeichen enthalten: Unicode-Zeichen, Ziffern, Leerzeichen sowie die folgenden Sonderzeichen: `_ . / = + - @`.

Einschränkungen für den Tag-Wert

- Tag-Werte müssen zwischen 0 und 255 Unicode-Zeichen lang sein.
- Tag-Werte können leer sein. Ansonsten müssen sie die folgenden Zeichen enthalten: Unicode-Zeichen, Ziffern, Leerzeichen und eines der folgenden Sonderzeichen: `_ . / = + - @`.

Taggen Sie Streams mit der Kinesis Data Streams Streams-Konsole

Mit der Kinesis Data Streams Streams-Konsole können Sie Tags zu Ihren Streams hinzufügen, aktualisieren, auflisten und entfernen.

Anzeigen der Tags für einen Stream

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Kinesis-Konsole unter https://console.aws.amazon.com/kinesis.
2. Wählen Sie im linken Navigationsbereich Datenstreams aus.
3. Wählen Sie auf der Seite Datenstreams den Stream aus, den Sie taggen möchten.
4. Wählen Sie auf der Seite mit den Stream-Details die Option Konfiguration aus.
5. Sehen Sie sich im Abschnitt „Tags“ die Tags an, die auf den Stream angewendet wurden.

Um einen Datenstream mit einem Tag zu erstellen

1. Öffnen Sie die Kinesis-Data-Streams-Konsole.
2. Wählen Sie im linken Navigationsbereich Datenströme aus.
3. Klicken Sie auf Create data stream (Daten-Stream erstellen).
4. Geben Sie auf der Seite Datenstream erstellen einen Namen für Ihren Datenstream ein.
5. Wählen Sie für die Kapazität des Datenstreams entweder den Modus On-Demand-Kapazität oder den Modus Bereitgestellte Kapazität aus.

   Weitere Informationen zu den Kapazitätsmodi finden Sie unterWählen Sie den richtigen Modus zum Streamen.
6. Gehen Sie im Abschnitt Tags wie folgt vor:

   a. Wählen Sie Neues Tag hinzufügen aus.

   b. Geben Sie unter Schlüssel das Tag ein und geben Sie optional einen Wert in das Feld Wert ein.

      Wenn Sie einen Fehler sehen, entspricht entweder der angegebene Tag-Schlüssel oder der Wert nicht den Tag-Einschränkungen. Weitere Informationen finden Sie unter Verstehen Sie die Einschränkungen von Tags.
7. Klicken Sie auf Create data stream (Daten-Stream erstellen).

Um ein Tag in einem Stream hinzuzufügen oder zu aktualisieren

1. Öffnen Sie die Kinesis-Data-Streams-Konsole.
2. Wählen Sie im linken Navigationsbereich Datenstreams aus.
3. Wählen Sie auf der Seite Datenstreams den Stream aus, zu dem Sie Tags hinzufügen oder aktualisieren möchten.
4. Wählen Sie auf der Seite mit den Stream-Details die Option Konfiguration aus.
5. Wählen Sie im Abschnitt Tags (Markierungen) die Option Manage tags (Tags (Markierungen) verwalten).
6. Führen Sie unter Tags einen der folgenden Schritte aus:

   - Um ein neues Tag hinzuzufügen, wählen Sie Neues Tag hinzufügen aus und geben Sie dann die Schlüssel - und Wertdaten des Tags ein. Wiederholen Sie diesen Schritt, bis Sie alle Tags hinzugefügt haben.

     Die maximale Anzahl von Tags, die Sie für jeden Stream hinzufügen können, ist 50.
   - Um ein vorhandenes Tag zu aktualisieren, geben Sie einen neuen Tag-Wert in das Feld Wert des Schlüssels dieses Tags ein.

   Wenn Sie einen Fehler sehen, entspricht entweder der angegebene Tag-Schlüssel oder der Wert nicht den Tag-Einschränkungen. Weitere Informationen finden Sie unter Verstehen Sie die Einschränkungen von Tags.
7. Wählen Sie Änderungen speichern aus.

Entfernen eines Tags von einem Stream

1. Öffnen Sie die Kinesis-Data-Streams-Konsole.
2. Wählen Sie im linken Navigationsbereich Datenstreams aus.
3. Wählen Sie auf der Seite Datenstreams den Stream aus, aus dem Sie Tags entfernen möchten.
4. Wählen Sie auf der Seite mit den Stream-Details die Option Konfiguration aus.
5. Wählen Sie im Abschnitt Tags (Markierungen) die Option Manage tags (Tags (Markierungen) verwalten).
6. Suchen Sie das Tag-Schlüssel-Wert-Paar, das Sie entfernen möchten. Wählen Sie dann Entfernen aus.

7. Wählen Sie Änderungen speichern aus.

Taggen Sie Streams mit dem AWS CLI

Mit dem können Sie Tags zu Ihren Streams hinzufügen, auflisten und entfernen AWS CLI. Beispiele finden Sie in der folgenden Dokumentation.

Stream erstellen

Erzeugt einen Stream mit Tags.

add-tags-to-stream

Fügt Tags für den angegebenen Stream hinzu oder aktualisiert diese.

list-tags-for-stream

Listet die Tags für den angegebenen Stream auf.

remove-tags-from-stream

Entfernt Tags von dem angegebenen Stream.

Taggen von Streams mithilfe der Kinesis Data Streams APIs

Mit den Kinesis Data Streams können Sie Tags zu Ihren Streams hinzufügen, auflisten und entfernen. APIs Beispiele finden Sie in der folgenden Dokumentation:

CreateStream

Erzeugt einen Stream mit Tags.

AddTagsToStream

Fügt Tags für den angegebenen Stream hinzu oder aktualisiert diese.

ListTagsForStream

Listet die Tags für den angegebenen Stream auf.

RemoveTagsFromStream

Entfernt Tags von dem angegebenen Stream.

Taggen Sie Verbraucher mit dem AWS CLI

Sie können Tags für Ihre Verbraucher hinzufügen, auflisten und entfernen, indem Sie die verwenden AWS CLI. Beispiele finden Sie in der folgenden Dokumentation:

register-stream-consumer

Registriert einen Verbraucher für einen Kinesis-Datenstrom mit Tags.

tag-resource

Fügt Tags für die angegebene Kinesis-Ressource hinzu oder aktualisiert sie.

list-tags-for-resource

Listet die Tags für die angegebene Kinesis-Ressource auf.

untag-resource

Entfernt Tags aus der angegebenen Kinesis-Ressource.

Taggen Sie Verbraucher, die Kinesis Data Streams verwenden APIs

Mit den Kinesis Data Streams APIs können Sie Tags für Ihre Verbraucher hinzufügen, auflisten und entfernen. Beispiele finden Sie in der folgenden Dokumentation:

RegisterStreamConsumer

Registriert einen Verbraucher für einen Kinesis-Datenstrom mit Tags.

TagResource

Fügt Tags für die angegebene Kinesis-Ressource hinzu oder aktualisiert sie.

ListTagsForResource

Listet die Tags für die angegebene Kinesis-Ressource auf.

UntagResource

Entfernt Tags aus der angegebenen Kinesis-Ressource.

Umgang mit großen Datensätzen

Amazon Kinesis Data Streams unterstützt Datensätze mit bis zu 10 Mebibyte (). MiBs Diese Funktion wird für die Verarbeitung intermittierender Datennutzlasten empfohlen, die die standardmäßige Datensatzgrößenbeschränkung von 1 MiB überschreiten. Die standardmäßige maximale Datensatzgröße für bestehende und neu erstellte Streams ist auf 1 MiB festgelegt.

Diese Funktion kommt IoT-Anwendungen (Internet of Things), CDC-Pipelines (Change Data Capture) und Workflows für maschinelles Lernen zugute, die gelegentlich die Verarbeitung größerer Datennutzlasten erfordern. Um mit der Verwendung großer Datensätze in deinem Stream zu beginnen, aktualisiere die maximale Datensatzgrößenbeschränkung für deinen Stream.

> **Important**
>
> Das individuelle Shard-Durchsatzlimit von 1 MB/s für Schreibvorgänge und 2 MB/s für Lesevorgänge bleibt unverändert, wobei größere Datensatzgrößen unterstützt werden. Kinesis Data Streams ist darauf ausgelegt, zeitweise große Datensätze zusammen mit einem Basisdatenverkehr von Datensätzen von weniger als oder gleich 1 MiB zu verarbeiten. Es ist nicht für die dauerhafte Aufnahme großer Datenmengen konzipiert.

Aktualisiere deinen Stream, um große Datensätze zu verwenden

Um größere Datensätze mit Kinesis Data Streams zu verarbeiten

1. Navigieren Sie zur Kinesis Data Streams Streams-Konsole.
2. Wählen Sie Ihren Stream aus und wechseln Sie zur Registerkarte Konfiguration.
3. Klicke neben Maximale Datensatzgröße auf Bearbeiten.
4. Stellen Sie Ihre maximale Datensatzgröße ein (bis zu 10 MiB).
5. Speichern Sie Ihre Änderungen.

Mit dieser Einstellung wird nur die maximale Datensatzgröße für diesen Kinesis-Datenstrom angepasst. Bevor Sie diesen Grenzwert erhöhen, stellen Sie sicher, dass alle Downstream-Anwendungen größere Datensätze verarbeiten können.

Sie können diese Einstellung auch über die AWS CLI aktualisieren:

```
aws kinesis update-max-record-size \ --stream-arn  \
        --max-record-size-in-ki-b 5000
```

Optimiere deine Stream-Leistung mit großen Datensätzen

Es wird empfohlen, große Datensätze auf weniger als 2% deines gesamten Traffics zu beschränken. In einem Stream hat jeder Shard eine Durchsatzkapazität von 1 MiB pro Sekunde. Um große Datensätze verarbeiten zu können, streamt Kinesis Data bis zu 10 Bursts MiBs, wobei der Durchschnitt bei 1 MiB pro Sekunde liegt. Diese Kapazität zur Unterstützung großer Datensätze wird kontinuierlich wieder in den Stream aufgenommen. Die Geschwindigkeit des Wiederauffüllens hängt von der Größe der großen Datensätze und der Größe des Basisdatensatzes ab. Die besten Ergebnisse erzielen Sie, wenn Sie einen gleichmäßig verteilten Partitionsschlüssel verwenden. Weitere Informationen zur Skalierung von Kinesis On-Demand-Modus finden Sie unter Funktionen und Anwendungsfälle für den On-Demand-Modus.

Reduzieren Sie die Drosselung bei großen Datensätzen

Um der Drosselung entgegenzuwirken

1. Implementieren Sie die Wiederholungslogik mit exponentiellem Back-off in Ihrer Producer-Anwendung.
2. Verwenden Sie zufällige Partitionsschlüssel, um große Datensätze auf verfügbare Shards zu verteilen.
3. Speichern Sie Payloads in Amazon S3 und senden Sie nur Metadatenverweise an den Stream für kontinuierliche Streams großer Datensätze. Weitere Informationen finden Sie unter Verarbeiten großer Datensätze mit Amazon Kinesis Data Streams.

Verarbeiten Sie große Datensätze mit den Kinesis Data Streams APIs

Die Unterstützung großer Datensätze führt eine neue API ein und aktualisiert zwei bestehende Steuerungsebenen, sodass bis APIs zu 10 MiBs Datensätze verarbeitet werden können.

API zum Ändern der Datensatzgröße:

- `UpdateMaxRecordSize`: Konfiguriert die maximale Datensatzgrößenbeschränkung für bestehende Streams auf bis zu 10. MiBs

Aktualisierungen APIs vorhandener:

- `CreateStream`: Fügt den optionalen `MaxRecordSizeInKiB` Parameter für die Festlegung von Größenbeschränkungen für Datensätze während der Stream-Erstellung hinzu.
- `DescribeStreamSummary`: Gibt das `MaxRecordSizeInKiB` Feld zurück, das die aktuelle Stream-Konfiguration anzeigt.

Alle APIs aufgelisteten Produkte behalten die Abwärtskompatibilität für bestehende Streams bei. Eine vollständige API-Dokumentation finden Sie in der Amazon Kinesis Data Streams Service API-Referenz.

AWS Komponenten, die mit großen Datensätzen kompatibel sind

Die folgenden AWS Komponenten sind mit großen Datensätzen kompatibel:

Komponente	Description
AWS SDK	AWS Das SDK unterstützt den Umgang mit großen Datensätzen. Sie können die maximale Datensatzgröße Ihres Streams auf bis zu 10 MiB aktualisieren, indem Sie die verfügbar en Methoden in der verwenden. AWS SDKs Weitere Informationen finden Sie unter Diesen Dienst mit einem AWS SDK verwenden.
Kinesis Consumer Library (KCL)	Ab Version 2.x unterstützt KCL die Verarbeit ung großer Datensätze. Um die Unterstützung großer Datensätze zu verwenden, aktualisi eren Sie `maxRecordSize` den Stream und verwenden Sie KCL. Weitere Informationen finden Sie unter Verwenden der Kinesis-C lientbibliothek.
Kinesis Producer-Bibliothek (KPL)	Ab Version 1.0.5 unterstützt KPL die Verarbeit ung großer Datensätze. Um die Unterstützung großer Datensätze zu verwenden, aktualisi eren Sie `maxRecordSize` den Stream und verwenden Sie KPL. Weitere Informationen

Komponente	Description
	finden Sie unter Entwickeln von Produzenten mithilfe der Amazon Kinesis Producer Library (KPL).
Amazon EMR	Amazon EMR mit Apache Spark unterstützt die Verarbeitung großer Datensätze bis zum Kinesis Data Streams Streams-Limit (10 MiBs). Verwenden Sie die Funktion, um die Unterstützung großer Datensätze zu verwenden. `readStream` Weitere Informationen finden Sie unter Integration von Amazon EMR und Amazon Kinesis.

Komponente	Description
Amazon Data Firehose	Bei Verwendung mit Kinesis Data Streams hängt das Verhalten von Amazon Data Firehose bei großen Datensätzen vom Lieferziel ab: • Amazon S3: Die Lieferung großer Datensätze wird ohne zusätzliche Konfiguration unterstützt. Wenn Sie die Datenformatkonvertierung verwenden, wird die Übermittlung großer Datensätze mit Firehose unterstützt. Wenn Sie dynamische Partitionierung verwenden, wird die Übermittlung großer Datensätze mit Firehose nicht unterstützt. • Lambda: Wir empfehlen nicht, große Datensätze mit Firehose zu verwenden, wenn dadurch Lambda-Funktionen im Downstream ausgelöst werden. Dies kann zu zeitweiligen Ausfällen führen. • HTTP: Die Lieferung großer Datensätze wird mit Firehose nicht unterstützt. • Snowflake: Die Lieferung großer Datensätze wird von Firehose nicht unterstützt. • Amazon Redshift: Die Lieferung großer Datensätze wird mit Firehose nicht unterstützt. Bei Anwendungen, die eine Lieferung an Snowflake oder Redshift mit großen Datensätzen erfordern, stellen Sie die Daten zuerst an Amazon S3 bereit. Verwenden Sie anschließend die ETL-Prozesse (Extract, Transform, Load), um die Daten zu laden. Testen Sie bei allen anderen Zielen das Verhalten mit

Komponente	Description
	großen Datensätzen in einer proof-of-concept Umgebung, bevor Sie die Skalierung auf die Produktionsnutzung durchführen. Der Umgang mit großen Datensätzen ist je nach Ziel unterschiedlich.
AWS Lambda	AWS Lambda unterstützt bis zu 6 MiBs Nutzlasten. Dieses Limit umfasst die Kinesis-Payload, die in die Base-64-Kodierung konvertiert wurde, und die Metadaten, die mit Event Source Mapping (ESM) verknüpft sind. Bei Datensätzen unter 6 verarbeitet Lambda sie mit ESM MiBs, ohne dass eine zusätzliche Konfiguration erforderlich ist. Bei Datensätzen, die größer als 6 sind MiBs, verarbeitet Lambda sie unter Verwendung eines Ziels für den Fall eines Fehlers. Sie müssen mithilfe von ESM ein Ziel für den Fall eines Fehlers konfigurieren, um Datensätze zu verarbeiten, die die Verarbeitungsgrenzen von Lambda überschreiten. Jedes Ereignis, das an das Ziel bei einem Fehler gesendet wird, ist ein JSON-Dokument, das Metadaten zum fehlgeschlagenen Aufruf enthält. Es wird empfohlen, unabhängig von der Datensatzgröße ein Ziel für den Fall eines Fehlers im ESM zu erstellen. Dadurch wird sichergestellt, dass keine Datensätze verworfen werden. Weitere Informationen finden Sie unter Ziele für fehlgeschlagene Aufrufe konfigurieren.

Komponente	Description
Amazon Redshift	Amazon Redshift unterstützt beim Streamen von Daten aus Kinesis Data Streams nur Datensatzgrößen von weniger als 1 MiB. Datensätze, die dieses Limit überschreiten, werden nicht verarbeitet. Datensätze, die nicht verarbeitet werden, werden protokolliert als `sys_stream_scan_errors` . Weitere Informationen finden Sie unter SYS_STREAM_SCAN_ERRORS.
Flink-Anschluss für Kinesis Data Streams	Es gibt zwei Ansätze für die Nutzung von Daten aus Kinesis Data Streams: den Kinesis-Quellkonnektor und den Kinesis-Sink-Connector. Der Quellanschluss unterstützt die Verarbeitung von Datensätzen mit weniger als 1 MiB und bis zu 10. MiBs Verwenden Sie den Senkenanschluss nicht für Aufzeichnungen, die größer als 1 MiB sind. Weitere Informationen finden Sie unter Verwenden von Konnektoren zum Verschieben von Daten in Amazon Managed Service für Apache Flink mit der DataStream API.

Regionen, in denen große Datensätze unterstützt werden

Diese Amazon Kinesis Data Streams Streams-Funktion ist nur in den folgenden AWS Regionen verfügbar:

AWS Region	Name der Region
eu-north-1	Europa (Stockholm)
me-south-1	Middle East (Bahrain)
ap-south-1	Asien-Pazifik (Mumbai)

AWS Region	Name der Region
eu-west-3	Europa (Paris)
ap-southeast-3	Asien-Pazifik (Jakarta)
us-east-2	USA Ost (Ohio)
af-south-1	Afrika (Kapstadt)
eu-west-1	Europa (Irland)
me-central-1	Naher Osten (VAE)
eu-central-1	Europa (Frankfurt)
sa-east-1	Südamerika (São Paulo)
ap-east-1	Asien-Pazifik (Hongkong)
ap-south-2	Asien-Pazifik (Hyderabad)
us-east-1	USA Ost (Nord-Virginia)
ap-northeast-2	Asien-Pazifik (Seoul)
ap-northeast-3	Asien-Pazifik (Osaka)
eu-west-2	Europa (London)
ap-southeast-4	Asien-Pazifik (Melbourne)
ap-northeast-1	Asien-Pazifik (Tokio)
us-west-2	USA West (Oregon)
us-west-1	USA West (Nordkalifornien)
ap-southeast-1	Asien-Pazifik (Singapur)
ap-southeast-2	Asien-Pazifik (Sydney)

AWS Region	Name der Region
il-central-1	Israel (Tel Aviv)
ca-central-1	Kanada (Zentral)
ca-west-1	Kanada West (Calgary)
eu-south-2	Europa (Spain)
cn-northwest-1	China (Ningxia)
eu-central-2	Europa (Zürich)
us-gov-east-1	AWS GovCloud (US-Ost)
us-gov-west-1	AWS GovCloud (US-West)

Führen Sie Resilienztests durch mit AWS Fault Injection Service

AWS Fault Injection Service ist ein vollständig verwalteter Service, der Sie bei der Durchführung von Fault-Injection-Experimenten für Ihre AWS Workloads unterstützt. AWS FIS Die Integration mit Amazon Kinesis Data Streams ermöglicht es Ihnen, die Widerstandsfähigkeit Ihrer Anwendung gegenüber häufigen Amazon Kinesis Data Streams Streams-API-Fehlern in einer kontrollierten Umgebung zu testen. Mit dieser Funktion können Sie die Fehlerbehandlung überprüfen, die Logik wiederholen und Systeme überwachen, bevor Fehler auftreten. Weitere Informationen finden Sie unter Was ist AWS Fault Injection Service? .

Aktionen

- Interner API-Fehler: Dadurch werden interne Fehler in Anfragen der IAM-Zielrolle eingefügt. Die spezifische Antwort hängt von jedem Dienst und jeder API ab. Die Aktion `aws:fis:inject-api-internal-error` erzeugt `InternalFailure` Fehler (HTTP 500).
- API-Drosselungsfehler: Dadurch werden interne Fehler in Anfragen der IAM-Zielrolle eingefügt. Die spezifische Antwort hängt von jedem Dienst und jeder API ab. Die Aktion `aws:fis:inject-api-throttle-error` erzeugt `ThrottlingException` Fehler (HTTP 400).
- Fehler „API nicht verfügbar“: Dadurch werden interne Fehler in Anfragen der IAM-Zielrolle eingefügt. Die spezifische Antwort hängt von jedem Dienst und jeder API ab. Die Aktion

`aws:fis:inject-api-unavailable-error` erzeugt `ServiceUnavailable` Fehler (HTTP 503).

- Von der API bereitgestellte Durchsatzausnahme: Dadurch werden interne Fehler in Anfragen der IAM-Zielrolle eingefügt. Die spezifische Antwort hängt von jedem Dienst und jeder API ab. Die Aktion `aws:kinesis:inject-api-provisioned-throughput-exception` erzeugt `ProvisionedThroughputExceededException` Fehler (HTTP 400).
- Ausnahme für abgelaufene API-Iteratoren: Dadurch werden interne Fehler in Anfragen eingefügt, die von der IAM-Zielrolle gestellt wurden. Die spezifische Antwort hängt von jedem Dienst und jeder API ab. Die Aktion `aws:kinesis:inject-api-expired-iterator-exception` erzeugt `ExpiredIteratorException` Fehler (HTTP 400).

Weitere Informationen finden Sie unter Amazon Kinesis Data Streams Streams-Aktionen.

Überlegungen

- Sie können die oben genannten Aktionen sowohl für bereitgestellte als auch für On-Demand-Angebote für Amazon Kinesis Data Streams verwenden.
- Ihr Streaming wird nach Abschluss des Experiments auf der Grundlage der ausgewählten Dauer fortgesetzt. Sie können ein laufendes Experiment auch beenden, bevor es abgeschlossen ist. Alternativ können Sie eine Stoppbedingung definieren, um das Experiment auf der Grundlage von Alarmen zu beenden, die den Zustand der Anwendung in einer Amazon CloudWatch Application Insights definieren.
- Sie können bis zu 280 Streams testen.

Weitere Informationen zum regionalen Support finden Sie unter AWS Fault Injection Service Endpunkte und Kontingente.

Ausnahmefehler beim bereitgestellten Durchsatz

Ausnahmefehler mit Überschreitung des bereitgestellten Durchsatzes (HTTP 400) treten auf, wenn die Anforderungsrate für einen Kinesis-Stream die Durchsatzgrenzen eines oder mehrerer Shards überschreitet. Jeder Shard hat spezifische Lese- und Schreibkapazitätsgrenzen, und eine Überschreitung dieser Grenzwerte löst diese Ausnahme aus. Zu den Szenarien, die zu dieser Ausnahme führen, gehören: plötzliche Spitzen bei der Datenaufnahme oder beim Datenverbrauch, unzureichende Shard-Kapazität für das zu verarbeitende Datenvolumen oder ungleichmäßige Verteilung der Partitionsschlüssel.

Empfehlungen für den Umgang mit Ausnahmen

- Implementieren Sie exponentielle Back-off- und Wiederholungsmechanismen.
- Erhöhen Sie die Anzahl der Shards, um einen höheren Durchsatz zu erreichen.
- Stellen Sie sicher, dass die Partitionsschlüssel ordnungsgemäß verteilt sind.
- Überwachen Sie die Stream-Metriken.

Darüber hinaus hilft die Verwendung des On-Demand-Kapazitätsmodus von Kinesis dabei, Workloads automatisch anzupassen und das Auftreten dieser Ausnahme zu minimieren. Weitere Informationen finden Sie unter Was ist AWS Fault Injection Service?

> **Note**
>
> Probleme mit unsachgemäßer Verteilung liegen außerhalb der Möglichkeiten der automatischen Skalierung im On-Demand-Modus.

Um ein einfaches Experiment durchzuführen

1. Verwenden Sie Basismetriken: Zeichnen Sie vor dem Testen normale Durchsatzmuster auf.
2. Erstellen Sie ein Experiment: Verwenden Sie die `aws:kinesis:inject-api-provisioned-throughput-exception` Aktion.
3. Konfigurieren Sie die Intensität: Beginnen Sie mit einer Anforderungsdrosselung von 25%.
4. Überwachen Sie die Antworten: Überprüfen Sie die Wiederholungslogik mit exponentiellem Back-off.
5. Überprüfen Sie die Skalierung: Vergewissern Sie sich, dass die auto-scaling die Aktivierung auslöst.
6. Überprüfen Sie die Alarme: Stellen Sie sicher, dass die `CloudWatch` Alarme wie erwartet laufen.

Die Anwendungen sollten geeignete Back-off-Strategien`WriteProvisionedThroughputExceeded`, Überwachungs- und `ReadProvisionedThroughputExceeded` Metriken implementieren und gegebenenfalls eine Shard-Skalierung auslösen.

Einzelheiten der Aktion

- Ressourcentyp: IAM-Rolle ARN
- Zieloperationen:`PutRecord`,, `PutRecords` `GetRecords`
- Fehlercode: `ProvisionedThroughputExceededException` (HTTP 400)
- Beschreibung: simuliert Szenarien, in denen die Anforderungsrate die Kapazitätsgrenzen der Shards überschreitet, und testet die Drosselung von Anwendungen und die Skalierung der Antworten.

Parameters

- IAM-Rolle ARN: Die Rolle, die Ihre Anwendung für Kinesis Data Streams Streams-Operationen verwendet.
- Operationen: Zieloperationen:`PutRecord`,,`PutRecords`. `GetRecords`
- Ressourcenliste: die spezifischen Stream-Namen oder Shard-Identifikatoren.
- Dauer: Die Versuchsdauer, d. h. die Dauer zwischen einer Minute und 12 Stunden. In der AWS FIS API ist der Wert eine Zeichenfolge im ISO 8601-Format. PT1M steht beispielsweise für eine Minute. In der AWS FIS Konsole geben Sie die Anzahl der Sekunden, Minuten oder Stunden ein.
- Intensität: Der Prozentsatz der Drosselungsanfragen.

Erforderliche Berechtigungen

- `kinesis:InjectApiError`

Beispiel für eine Versuchsvorlage

Das folgende Beispiel zeigt eine bereitgestellte Durchsatzausnahme für alle Anfragen bis zu 5 Kinesis Data-Streams mit dem angegebenen Tag. AWS FIS wählt die Streams, die beeinflusst werden sollen, nach dem Zufallsprinzip aus. Nach 5 Minuten ist der Fehler behoben.

```
{
    "description": "Kinesis stream experiment",
    "targets": {
        "KinesisStreams-Target-1": {
            "resourceType": "aws:kinesis:stream",
            "resourceTags": {
```

```
                    "tag-key": "tag-value"
                },
                "selectionMode": "COUNT(5)"
            }
        },
        "actions": {
             "kinesis": {
                  "actionId": "aws:kinesis:stream-provisioned-throughput-exception",
                  "description": "my-stream",
                  "parameters": {
                       "duration": "PT5M",
                       "percentage": "100",
                       "service": "kinesis"
                  },
                  "targets": {
                        "KinesisStreams": "KinesisStreams-Target-1"
                  }
             }
    },
    "stopConditions": [
             {
                  "source": "none"
             }
    ],
    "roleArn": "arn:aws:iam::111122223333:role/role-name",
    "tags": {},
    "experimentOptions": {
        "accountTargeting": "single-account",
        "emptyTargetResolutionMode": "fail"
    }
}
```

Beispiel für ein Experiment, Rollenberechtigungen

Mit der folgenden Berechtigung können Sie die `aws:kinesis:stream-expired-iterator-exception` Aktionen `aws:kinesis:stream-provisioned-throughput-exception` und für einen bestimmten Stream ausführen, die sich auf 50% der Anfragen auswirken.

Exception-Fehler im abgelaufenen Iterator

Ausnahmefehler bei abgelaufenen Iteratoren (HTTP 400) treten auf, wenn der Shard-Iterator abgelaufen ist und beim Aufruf nicht mehr zum Abrufen von Stream-Datensätzen verwendet wird. `GetRecords` Dies tritt auf, wenn es zu Verzögerungen zwischen Lesevorgängen kommt, die durch

lang andauernde Datenverarbeitungsaufgaben, Netzwerkprobleme oder Anwendungsausfälle verursacht werden.

> **Note**
>
> Ein Shard-Iterator ist ab dem Zeitpunkt seiner Ausgabe noch 5 Minuten gültig.

Empfehlungen für den Umgang mit Ausnahmen

- Shard-Iteratoren werden aktualisiert, bevor sie ablaufen.
- Integration der Fehlerbehandlung, um neue Iteratoren zu erhalten.
- Verwendung der Kinesis Kinesis Client Library (KCL), die den Ablauf des Shard-Iterators automatisch verwaltet.

Weitere Informationen finden Sie unter Was ist? AWS Fault Injection Service

Um ein einfaches Experiment durchzuführen

1. Erstellen Sie eine Experimentvorlage: Verwenden Sie die AWS FIS Konsole.
2. Wählen Sie die Aktion aus: Verwenden Sie die `aws:kinesis:inject-api-expired-iterator-exception` Aktion.
3. Konfigurieren Sie die Ziele: Geben Sie die IAM-Rolle und die Kinesis Data Streams-Operationen an.
4. Legen Sie die Dauer fest: Beginnen Sie mit 5-10 Minuten für die ersten Tests.
5. Stoppbedingungen hinzufügen: Stoppbedingungen für AWS FIS.
6. Führen Sie das Experiment aus: Überwachen Sie das Verhalten der Anwendung.

Einzelheiten zur Aktion

- Ressourcentyp: IAM-Rolle ARN
- Zieloperationen: `GetRecords`
- Fehlercode: `ExpiredIteratorException` (HTTP 400)
- Beschreibung: Der bereitgestellte Iterator überschreitet das zulässige Höchstalter und simuliert Szenarien, in denen die Verarbeitung von Datensätzen zu langsam ist oder die Checkpoint-Logik fehlschlägt.

Parameters

- IAM-Rolle ARN: Die Rolle, die Ihre Anwendung für Kinesis Data Streams Streams-Operationen verwendet.
- Operationen: Zieloperationen: `GetRecords`
- Ressourcenliste: die spezifischen Stream-Namen oder ARNs.
- Dauer: Die Dauer des Experiments. Dies ist konfigurierbar.
- Intensität: Der Prozentsatz der Drosselungsanfragen.

Erforderliche Berechtigungen

- `kinesis:InjectApiError`

Daten in Amazon Kinesis Data Streams schreiben

Ein Produzent ist eine Anwendung, die Daten in Amazon Kinesis Data Streams schreibt. Sie können mit der AWS SDK für Java und der Kinesis Producer Library (KPL) Producer Library (KPL) Producer für Kinesis Data Streams erstellen.

Wenn Sie Kinesis Data Streams zum ersten Mal verwenden, sollten Sie sich zunächst mit den Konzepten und der Terminologie in Was ist Amazon Kinesis Data Streams? und Verwenden Sie die AWS CLI , um Amazon Kinesis Data Streams Streams-Operationen durchzuführen vertraut machen.

> **Important**
>
> Kinesis Data Streams unterstützt Änderungen des Zeitraums der Datensatzaufbewahrung für einen Datenstrom. Weitere Informationen finden Sie unter Ändern Sie den Aufbewahrungszeitraum für Daten.

Zur Übergabe von Daten an einen Stream müssen Sie den Namen des Streams, einen Partitionsschlüssel und den Daten-Blob, der zum Stream hinzugefügt wird, angeben. Der Partitionsschlüssel wird verwendet, um zu ermitteln, zu welchem Shard im Stream der Datensatz hinzugefügt wird.

Alle Daten im Shard werden an denselben Worker gesendet, der den Shard verarbeitet. Welchen Partitionsschlüssel Sie verwenden, hängt von Ihrer Anwendungslogik ab. Die Anzahl der Partitionsschlüssel sollte in der Regel viel größer sein als die Anzahl der Shards. Der Grund hierfür ist, dass über den Partitionsschlüssel festgelegt wird, wie ein Datensatz einem bestimmten Shard zugeordnet wird. Wenn Sie genügend Partitionsschlüssel haben, können die Daten gleichmäßig auf die Shards in einem Stream verteilt werden.

Topics

- Entwickeln Sie Produzenten mithilfe der Amazon Kinesis Producer Library (KPL)
- Entwickeln Sie Produzenten mithilfe der Amazon Kinesis Data Streams Streams-API mit dem AWS SDK für Java
- Schreiben Sie mit Kinesis Agent in Amazon Kinesis Data Streams
- Schreiben Sie mithilfe anderer AWS Dienste in Kinesis Data Streams
- Schreiben Sie mithilfe von Integrationen von Drittanbietern in Kinesis Data Streams

- Problembehandlung Amazon Kinesis Data Streams Streams-Produzenten
- Optimieren Sie Kinesis Data Streams Streams-Produzenten

Entwickeln Sie Produzenten mithilfe der Amazon Kinesis Producer Library (KPL)

Ein Produzent von Amazon Kinesis Data Streams ist eine Anwendung, die Datensätze an einen Kinesis-Datenstrom übergibt (dies wird auch als Datenerfassung bezeichnet). Die Amazon Kinesis Producer Library (KPL) vereinfacht die Entwicklung von Herstelleranwendungen und ermöglicht es Entwicklern, einen hohen Schreibdurchsatz in einen Kinesis-Datenstrom zu erzielen.

Sie können die KPL mit Amazon CloudWatch überwachen. Weitere Informationen finden Sie unter Überwachen Sie die Kinesis Producer Library mit Amazon CloudWatch.

Topics

- Sehen Sie sich die Rolle der KPL an
- Machen Sie sich die Vorteile der Verwendung der KPL bewusst
- Verstehen Sie, wann Sie die KPL nicht verwenden sollten
- Installieren Sie die KPL
- Migrieren Sie von KPL 0.x zu KPL 1.x
- Umstellung auf Amazon Trust Services (ATS) -Zertifikate für die KPL
- Von KPL unterstützte Plattformen
- Die wichtigsten Konzepte von KPL
- Integrieren Sie die KPL in den Herstellercode
- Schreiben Sie mit der KPL in Ihren Kinesis-Datenstream
- Konfiguration der Amazon Kinesis Producer Library
- Implementieren Sie die Deaggregation für Verbraucher
- Verwenden Sie die KPL mit Amazon Data Firehose
- Verwenden Sie die KPL mit der Schema Registry AWS Glue
- Konfigurieren Sie die KPL-Proxykonfiguration
- Richtlinie zum Lebenszyklus der KPL-Version

> **Note**
>
> Es wird empfohlen, ein Upgrade auf die neueste KPL-Version durchzuführen. KPL wird regelmäßig mit neueren Versionen aktualisiert, die die neuesten Abhängigkeits- und Sicherheitspatches, Bugfixes und abwärtskompatible neue Features enthalten. Weitere Informationen finden Sie unter https://github.com/awslabs/amazon-kinesis-producer/releases/.

Sehen Sie sich die Rolle der KPL an

Die KPL ist eine hochgradig konfigurierbare Bibliothek easy-to-use, mit der Sie in einen Kinesis-Datenstrom schreiben können. Sie dient als Vermittler zwischen dem Code Ihrer Produzentenanwendung und den API-Aktionen von Kinesis Data Streams. Die KPL führt die folgenden primären Aufgaben durch:

- Schreiben in einen oder mehrere Kinesis-Datenströme mit einem automatischen und konfigurierbaren Wiederholungsmechanismus
- Sammeln von Datensätzen und Verwenden von `PutRecords`, um pro Anforderung mehrere Datensätze für mehrere Shards zu schreiben
- Aggregieren von Benutzerdatensätzen zur Erhöhung der Nutzlast und Verbesserung des Durchsatzes
- Nahtlose Integration in die Kinesis Client Library (KCL), für eine Disaggregation gestapelter Datensätze auf Konsumentenseite
- Übermittelt in Ihrem Namen CloudWatch Amazon-Metriken, um Einblick in die Leistung der Produzenten zu erhalten

Beachten Sie, dass sich die KPL von der Kinesis Data Streams Streams-API unterscheidet, die in der verfügbar ist. AWS SDKs Die API für Kinesis Data Streams unterstützt Sie bei der Verwaltung vieler Aspekte von Kinesis Data Streams (einschließlich Stream-Erstellung, Resharding sowie Hinzufügen und Abrufen von Datensätzen), während die KPL eine Abstraktionsebene speziell für die Datenübernahme bereitstellt. Informationen zur Kinesis-Data-Streams-API finden Sie in der Amazon-Kinesis-API-Referenz.

Machen Sie sich die Vorteile der Verwendung der KPL bewusst

In der folgenden Liste sind einige der größten Vorteile der Verwendung der KPL für die Entwicklung von Produzenten von Kinesis Data Streams aufgeführt.

Die KPL kann für synchrone oder asynchrone Anwendungsfälle eingesetzt werden. Wir empfehlen, die höhere Leistung der asynchronen Schnittstelle zu verwenden, sofern keine besonderen Gründe für die Nutzung des synchronen Verhaltens vorliegen. Weitere Informationen zu diesen beiden Anwendungsfällen sowie einen Beispiel-Code finden Sie unter Schreiben Sie mit der KPL in Ihren Kinesis-Datenstream.

Leistungsvorteile

Mit der KPL können Sie leistungsstarke Produzenten erstellen. Angenommen, Ihre Amazon-EC2-Instances dienen als Proxy zum Erfassen von 100-Byte-Ereignissen von mehreren Hundert oder Tausend energiesparenden Geräten und zum Schreiben von Datensätzen in einen Kinesis-Datenstrom. Diese EC2 Instances müssen jeweils Tausende von Ereignissen pro Sekunde in Ihren Datenstream schreiben. Um den nötigen Durchsatz zu erreichen, müssen Produzenten eine komplexe Logik, beispielsweise Stapelverarbeitung oder Multithreading, implementieren und auf Konsumentenseite eine Wiederholung der Logik sowie die Disaggregation der Datensätze sicherstellen. Die KPL führt all diese Aufgaben für Sie durch.

Anwenderfreundlichkeit auf Konsumentenseite

Entwickler auf Konsumentenseite, die die KCL in Java nutzen, können die KPL ohne zusätzlichen Aufwand integrieren. Wenn die KCL einen aggregierten Datensatz von Kinesis Data Streams abruft, der aus mehreren KPL-Benutzerdatensätzen besteht, ruft sie automatisch die KPL auf, um die einzelnen Benutzerdatensätze vor der Rückgabe an den Benutzer zu extrahieren.

Für Entwickler auf Verbraucherseite, die anstelle der KCL direkt die API-Operation `GetRecords` nutzen, steht eine KPL-Java-Bibliothek zum Extrahieren der einzelnen Benutzerdatensätze vor der Rückgabe an den Benutzer zur Verfügung.

Überwachen von Produzenten

Sie können Ihre Kinesis Data Streams Streams-Produzenten mithilfe von Amazon und der KPL sammeln, überwachen CloudWatch und analysieren. Die KPL sendet in Ihrem Namen Durchsatz-, Fehler- und andere Messwerte an und kann für die Überwachung CloudWatch auf Stream-, Shard- oder Producer-Ebene konfiguriert werden.

Asynchrone Architektur

Da die KPL Datensätze zwischenspeichern kann, bevor sie an Kinesis Data Streams gesendet werden, zwingt sie die aufrufende Anwendung nicht, zu blockieren und auf eine Bestätigung zu warten, dass der Datensatz auf dem Server angekommen ist, bevor die Laufzeit fortgesetzt wird. Der Aufruf, einen Datensatz an die KPL zu übergeben, wird stets sofort zurückgegeben. Es wird nicht darauf gewartet, dass der Datensatz gesendet bzw. eine Antwort vom Server empfangen wird. Stattdessen wird ein `Future`-Objekt erstellt, das zu einem späteren Zeitpunkt das Ergebnis der Sendung des Datensatzes an Kinesis Data Streams empfängt. Dies ist dasselbe Verhalten wie bei asynchronen Clients im SDK. AWS

Verstehen Sie, wann Sie die KPL nicht verwenden sollten

Die KPL kann eine weitere Verarbeitungsverzögerung von `RecordMaxBufferedTime` innerhalb der Bibliothek verursachen (vom Benutzer konfigurierbar). Größere Werte von `RecordMaxBufferedTime` führen zu einer schnelleren Verpackung und einer besseren Leistung. Anwendungen, die diese zusätzliche Verzögerung nicht tolerieren können, müssen das AWS SDK möglicherweise direkt verwenden. Weitere Informationen zur Verwendung des AWS SDK mit Kinesis Data Streams finden Sie unterEntwickeln Sie Produzenten mithilfe der Amazon Kinesis Data Streams Streams-API mit dem AWS SDK für Java. Weitere Informationen zu `RecordMaxBufferedTime` und anderen vom Benutzer konfigurierbaren Eigenschaften der KPL finden Sie unter Konfiguration der Amazon Kinesis Producer Library.

Installieren Sie die KPL

Amazon stellt vorgefertigte Binärdateien der C++ Amazon Kinesis Producer Library (KPL) für macOS, Windows und aktuelle Linux-Distributionen bereit (Einzelheiten zu den unterstützten Plattformen finden Sie im nächsten Abschnitt). Diese Binärdateien sind Teil der JAR-Dateien von Java und werden automatisch aufgerufen und verwendet, wenn Sie das Paket mit Maven installieren. Mit den folgenden Maven-Links können Sie nach den neusten Versionen der KPL und KCL suchen:

- KPL
- KCL

Die Linux-Binärdateien wurden mit der GNU Compiler Collection (GCC) kompiliert und statisch mit libstdc++ unter Linux verknüpft. Sie werden voraussichtlich von allen 64-Bit-Linux-Distributionen unterstützt, in denen glibc in der Version 2.5 oder höher enthalten ist.

Benutzer früherer Linux-Distributionen können die KPL mithilfe der Build-Anweisungen erstellen, die zusammen mit dem Quellcode bereitgestellt werden. GitHub Informationen zum Herunterladen der KPL finden Sie in der GitHub Amazon Kinesis Producer Library.

> **Important**
>
> Amazon Kinesis Producer Library (KPL) 0.x wird end-of-support am 30. Januar 2026 verfügbar sein. Wir empfehlen dringend, dass Sie Ihre KPL-Anwendungen, die Version 0.x verwenden, vor dem 30. Januar 2026 auf die neueste KPL-Version migrieren. Die neueste KPL-Version finden Sie auf der KPL-Seite auf Github. Informationen zur Migration von KPL 0.x zu KPL 1.x finden Sie unter. Migrieren Sie von KPL 0.x zu KPL 1.x

Migrieren Sie von KPL 0.x zu KPL 1.x

Dieses Thema enthält step-by-step Anweisungen zur Migration Ihres Verbrauchers von KPL 0.x zu KPL 1.x. KPL 1.x bietet Unterstützung für AWS SDK für Java 2.x unter Beibehaltung der Schnittstellenkompatibilität mit früheren Versionen. Sie müssen Ihre zentrale Datenverarbeitungslogik nicht aktualisieren, um auf KPL 1.x zu migrieren.

1. Stellen Sie sicher, dass Sie die folgenden Voraussetzungen erfüllen:
   - Java Development Kit (JDK) 8 oder höher
   - AWS SDK für Java 2.x
   - Maven oder Gradle für das Abhängigkeitsmanagement
2. Fügen Sie Abhängigkeiten hinzu

   Wenn Sie Maven verwenden, fügen Sie Ihrer Datei pom.xml die folgende Abhängigkeit hinzu. Stellen Sie sicher, dass Sie die groupId von `com.amazonaws` bis `software.amazon.kinesis` und die Version `1.x.x` auf die neueste KPL-Version aktualisiert haben.

   ```
   <dependency>
       <groupId>software.amazon.kinesis</groupId>
       <artifactId>amazon-kinesis-producer</artifactId>
       <version>1.x.x</version> <!-- Use the latest version -->
   </dependency>
   ```

Wenn Sie Gradle verwenden, fügen Sie Ihrer Datei Folgendes hinzu. `build.gradle` Stellen Sie sicher, dass Sie es `1.x.x` durch die neueste KPL-Version ersetzen.

```
implementation 'software.amazon.kinesis:amazon-kinesis-producer:1.x.x'
```

Sie können im Maven Central Repository nach der neuesten Version der KPL suchen.

3. Aktualisieren Sie die Importanweisungen für KPL

KPL 1.x verwendet AWS SDK für Java 2.x und verwendet einen aktualisierten Paketnamen, der mit beginnt`software.amazon.kinesis`, verglichen mit dem Paketnamen in der vorherigen KPL, der mit beginnt. `com.amazonaws.services.kinesis`

Ersetzen Sie den Import für durch. `com.amazonaws.services.kinesis` `software.amazon.kinesis` In der folgenden Tabelle sind die Importe aufgeführt, die Sie ersetzen müssen.

Importiere Ersatzprodukte

Ersetzen Sie:	mit:
com.amazonaws.services.kinesis.producer.Attempt importieren;	software.amazon.kinesis.producer.Attempt importieren;
importiere com.amazonaws.services.kinesis.producer. BinaryToHexConverter;	software.amazon.kinesis.producer importieren. BinaryToHexConverter;
importiere com.amazonaws.services.kinesis.producer. CertificateExtractor;	software.amazon.kinesis.producer importieren. CertificateExtractor;
com.amazonaws.services.kinesis.producer.Daemon importieren;	software.amazon.kinesis.producer.Daemon importieren;
com.amazonaws.services.kinesis.producer importieren. DaemonException;	software.amazon.kinesis.producer importieren. DaemonException;
importiere com.amazonaws.services.kinesis.producer. FileAgeManager;	software.amazon.kinesis.producer importieren. FileAgeManager;

Ersetzen Sie:	mit:
importiere com.amazonaws.services.kinesis.producer. FutureTimedOutException;	software.amazon.kinesis.producer importieren. FutureTimedOutException;
importiere com.amazonaws.services.kinesis.producer. GlueSchemaRegistrySerializerInstance;	software.amazon.kinesis.producer importieren. GlueSchemaRegistrySerializerInstance;
importiere com.amazonaws.services.kinesis.producer. HashedFileCopier;	software.amazon.kinesis.producer importieren. HashedFileCopier;
importiere com.amazonaws.services.kinesis.producer. IKinesisProduzent;	software.amazon.kinesis.producer importieren. IKinesisProduzent;
importiere com.amazonaws.services.kinesis.producer. IrrecoverableError;	software.amazon.kinesis.producer importieren. IrrecoverableError;
importiere com.amazonaws.services.kinesis.producer. KinesisProducer;	software.amazon.kinesis.producer importieren. KinesisProducer;
importiere com.amazonaws.services.kinesis.producer. KinesisProducerConfiguration;	software.amazon.kinesis.producer importieren. KinesisProducerConfiguration;
importiere com.amazonaws.services.kinesis.producer. LogInputStreamReader;	software.amazon.kinesis.producer importieren. LogInputStreamReader;
com.amazonaws.services.kinesis.producer.Metric importieren;	software.amazon.kinesis.producer.Metric importieren;
com.amazonaws.services.kinesis.producer importieren. ProcessFailureBehavior;	software.amazon.kinesis.producer importieren. ProcessFailureBehavior;
importiere com.amazonaws.services.kinesis.producer. UnexpectedMessageException;	software.amazon.kinesis.producer importieren. UnexpectedMessageException;
importiere com.amazonaws.services.kinesis.producer. UserRecord;	software.amazon.kinesis.producer importieren. UserRecord;

Ersetzen Sie:	mit:
importiere com.amazonaws.services.kinesis.producer. UserRecordFailedException;	software.amazon.kinesis.producer importieren. UserRecordFailedException;
importiere com.amazonaws.services.kinesis.producer. UserRecordResult;	software.amazon.kinesis.producer importieren. UserRecordResult;
com.amazonaws.services.kinesis.producer.protobuf.Messages importieren;	software.amazon.kinesis.producer.protobuf.Messages importieren;
com.amazonaws.services.kinesis.producer.protobuf.config importieren;	software.amazon.kinesis.producer.protobuf.config importieren;

4. Importanweisungen für Anbieterklassen für Anmeldeinformationen aktualisieren AWS

   Bei der Migration zu KPL 1.x müssen Sie Pakete und Klassen in Ihren Importen in Ihrem KPL-Anwendungscode, die auf der Version AWS SDK für Java 1.x basieren, auf die entsprechenden Pakete und Klassen aktualisieren, die auf der Version 2.x basieren. AWS SDK für Java Übliche Importe in der KPL-Anwendung sind Anbieterklassen für Anmeldeinformationen. Eine vollständige Liste der Änderungen des Anmeldeinformationsanbieters finden Sie in der Dokumentation zum AWS SDK für Java 2.x-Migrationshandbuch unter Änderungen des Anmeldeinformationsanbieters. Im Folgenden finden Sie die häufigsten Importänderungen, die Sie möglicherweise in Ihren KPL-Anwendungen vornehmen müssen.

   Importieren Sie in KPL 0.x

   ```
   import com.amazonaws.auth.DefaultAWSCredentialsProviderChain;
   ```

   In KPL 1.x importieren

   ```
   import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
   ```

   Wenn Sie andere Anbieter für Anmeldeinformationen importieren, die auf Version AWS SDK für Java 1.x basieren, müssen Sie sie auf die entsprechenden Anbieter von AWS SDK für Java 2.x aktualisieren. Wenn Sie keine classes/packages aus Version AWS SDK für Java 1.x importiert haben, können Sie diesen Schritt ignorieren.

5. Aktualisieren Sie die Konfiguration des Anmeldeinformationsanbieters in der KPL-Konfiguration

Die Konfiguration des Anmeldeinformationsanbieters in KPL 1.x erfordert die Anmeldeinformationsanbieter AWS SDK für Java 2.x. Wenn Sie die Anbieter für Anmeldeinformationen für die Version AWS SDK für Java 1.x in der übergeben, `KinesisProducerConfiguration` indem Sie den standardmäßigen Anbieter für Anmeldeinformationen überschreiben, müssen Sie ihn mit den Anmeldeinformationsanbietern der Version 2.x aktualisieren. AWS SDK für Java Eine vollständige Liste der Änderungen des Anmeldeinformationsanbieters finden Sie in der Dokumentation zum Migrationshandbuch für AWS SDK für Java 2.x unter Änderungen des Anmeldeinformationsanbieters. Wenn Sie den standardmäßigen Anbieter für Anmeldeinformationen in der KPL-Konfiguration nicht überschrieben haben, können Sie diesen Schritt ignorieren.

Wenn Sie beispielsweise den standardmäßigen Anbieter für Anmeldeinformationen für die KPL mit dem folgenden Code überschreiben:

```
KinesisProducerConfiguration config = new KinesisProducerConfiguration();
// SDK v1 default credentials provider
config.setCredentialsProvider(new DefaultAWSCredentialsProviderChain());
```

Sie müssen sie mit dem folgenden Code aktualisieren, um den AWS SDK für Java 2.x-Anmeldeinformationsanbieter verwenden zu können:

```
KinesisProducerConfiguration config = new KinesisProducerConfiguration();
// New SDK v2 default credentials provider
config.setCredentialsProvider(DefaultCredentialsProvider.create());
```

Umstellung auf Amazon Trust Services (ATS) -Zertifikate für die KPL

Am 9. Februar 2018 um 9:00 AM PST hat Amazon Kinesis Data Streams ATS-Zertifikate installiert. Um weiterhin Datensätze mithilfe der Amazon Kinesis Producer Library (KPL) in Kinesis Data Streams schreiben zu können, müssen Sie Ihre Installation der KPL auf Version 0.12.6 oder höher aktualisieren. Diese Änderung betrifft alle Regionen. AWS

Informationen zur Umstellung auf ATS finden Sie unter So bereiten Sie sich auf die AWS Umstellung auf eine eigene Zertifizierungsstelle vor.

Falls Sie Probleme haben und technischen Support benötigen, erstellen Sie einen Fall im AWS Support Center.

Von KPL unterstützte Plattformen

Die Amazon Kinesis Producer Library (KPL) ist in C++ geschrieben und wird als untergeordneter Prozess des Hauptbenutzerprozesses ausgeführt. Vorkompilierte native 64-Bit-Binärdateien sind in der Java-Version enthalten und werden vom Java-Wrapper verwaltet.

Das Java-Paket kann auf folgenden Betriebssystemen ausgeführt werden, ohne dass zusätzliche Bibliotheken installiert werden müssen:

- Linux-Distributionen mit Kernel 2.6.18 (September 2006) und höher
- Apple iOS X 10.9 und höher
- Windows Server 2008 und höher

> **Important**
>
> Windows Server 2008 und höher wird für alle KPL-Versionen bis Version 0.14.0 unterstützt. Die Windows-Plattform wird ab KPL Version 0.14.0 oder höher NICHT unterstützt.

Beachten Sie, dass es sich bei der KPL um eine 64-Bit-Version handelt.

Quellcode

Für den Fall, dass die Binärdateien, die bei der KPL-Installation bereitgestellt werden, für Ihre Umgebung nicht ausreichen, wird der Kern der KPL als C++-Modul bereitgestellt. Der Quellcode für das C++-Modul und die Java-Schnittstelle sind unter der Amazon Public License veröffentlicht und GitHub in der Amazon Kinesis Producer Library verfügbar. Obwohl die KPL auf allen Plattformen verwendet werden kann, für die ein neuer, standardmäßiger C++-Compiler und eine Java-Laufzeitumgebung verfügbar sind, unterstützt Amazon offiziell keine Plattformen, die nicht auf der Liste der unterstützten Plattformen enthalten sind.

Die wichtigsten Konzepte von KPL

Die folgenden Abschnitte enthalten Konzepte und Terminologie, die Sie benötigen, um die Amazon Kinesis Producer Library (KPL) zu verstehen und davon zu profitieren.

Themen

- Datensätze
- Stapelverarbeitung

- Aggregation
- Sammlung

Datensätze

In dieser Anleitung wird zwischen KPL-Benutzerdatensätzen und Datensätzen von Kinesis Data Streams unterschieden. Wenn von einem Datensatz ohne weitere Angabe die Rede ist, ist ein KPL-Benutzerdatensatz gemeint. Wenn wir uns auf einen Datensatz von Kinesis Data Streams beziehen, sagen wir ausdrücklich Datensatz von Kinesis Data Streams.

Ein KPL-Benutzerdatensatz ist ein Daten-Blob mit einer bestimmten Bedeutung für den Benutzer. Zu den Beispielen zählen ein JSON-Blob, der ein Ereignis auf der Benutzeroberfläche einer Website darstellt oder der Protokolleintrag eines Webservers.

Ein Datensatz von Kinesis Data Streams ist eine Instance der `Record`-Datenstruktur, die von der API des Kinesis-Data-Streams-Dienstes definiert wird. Er enthält eine Sequenznummer, einen Partitionsschlüssel und einen Daten-Blob.

Stapelverarbeitung

Unter Stapelverarbeitung versteht man das Durchführen einer einzelnen Aktion für mehrere Elemente gleichzeitig (im Gegensatz zur Ausführung ein und derselben Aktion für jedes Element separat).

In diesem Kontext steht „Element“ für einen Datensatz, der durch die Aktion an Kinesis Data Streams gesendet wird. Ohne Stapelverarbeitung würden Sie jeden Datensatz in einen separaten Datensatz von Kinesis Data Streams platzieren und eine HTTP-Anforderung erstellen, um ihn an Kinesis Data Streams zu senden. Mit Stapelverarbeitung können mehrere Datensätze in eine HTTP-Anforderung eingeschlossen werden.

Die KPL unterstützt zwei Arten der Stapelverarbeitung:

- Aggregation – Speichern mehrerer Datensätze in einem einzelnen Datensatz von Kinesis Data Streams.
- Sammlung – Verwenden der API-Operation `PutRecords` zum Senden mehrerer Datensätze aus Kinesis Data Streams an einen oder mehrere Shards in Ihrem Kinesis-Datenstrom.

Die zwei Arten der KPL-Stapelverarbeitung sind so konzipiert, dass sie gleichzeitig und unabhängig voneinander aktiviert oder deaktiviert werden können. Standardmäßig sind beide aktiviert.

Aggregation

Aggregation bezieht sich auf die Speicherung mehrerer Datensätze in einem Datensatz von Kinesis Data Streams. Die Aggregation ermöglicht Kunden, die Anzahl der Datensätze zu erhöhen, die pro API-Aufruf gesendet werden. Dadurch steigt der Durchsatz des Produzenten deutlich.

Shards von Kinesis Data Streams unterstützen bis zu 1 000 Datensätze von Kinesis Data Streams pro Sekunde bzw. einen Durchsatz von 1 MB. Das Limit für Datensätze pro Sekunde von Kinesis Data Streams bindet Kunden mit Datensätzen kleiner als 1 KB. Durch die Datensatzaggregation können Kunden mehrere Datensätze in einem einzelnen Datensatz von Kinesis Data Streams zusammenfassen. So können Kunden Ihren Durchsatz pro Shard verbessern.

Stellen Sie sich den Fall eines Shards in der Region us-east-1 vor, der derzeit mit einer konstanten Rate von 1.000 Datensätzen pro Sekunde mit Datensätzen von jeweils 512 Byte ausgeführt wird. Mit der KPL-Aggregation können Sie 1 000 Datensätze in nur 10 Datensätze von Kinesis Data Streams verpacken und reduzieren so die Anfragen pro Sekunde (RPS) auf 10 (jeweils 50 KB).

Sammlung

Mit Sammlung ist die Stapelverarbeitung mehrerer Datensätze von Kinesis Data Streams und das Senden von diesen in einer einzelnen HTTP-Anforderung mit einem Aufruf der API-Operation `PutRecords` gemeint. Dies steht im Gegensatz zum Senden einzelner Datensätze von Kinesis Data Streams in separaten HTTP-Anforderungen.

Durch eine Sammlung wird der Durchsatz erhöht, da der Aufwand für das Erstellen vieler separater HTTP-Anforderungen entfällt. `PutRecords` wurde speziell für diesen Zweck entwickelt.

Eine Sammlung unterscheidet sich von der Aggregation dahingehend, dass mit Gruppen von Datensätzen von Kinesis Data Streams gearbeitet wird. Die gesammelten Datensätze von Kinesis Data Streams können trotzdem mehrere Datensätze eines Benutzers enthalten. Die Beziehung lässt sich wie folgt visualisieren:

```
record 0 --|
record 1   |        [ Aggregation ]
    ...    |--> Amazon Kinesis record 0 --|
    ...    |                              |
record A --|                              |
                                           |
    ...                  ...               |
                                           |
record K --|                              |
```

```
record L   |                                |      [ Collection ]
    ...    |--> Amazon Kinesis record C --|--> PutRecords Request
    ...    |                                |
record S --|                                |
                                            |
    ...                  ...                |
                                            |
record AA--|                                |
record BB  |                                |
    ...    |--> Amazon Kinesis record M --|
    ...    |
record ZZ--|
```

Integrieren Sie die KPL in den Herstellercode

Die Amazon Kinesis Producer Library (KPL) wird in einem separaten Prozess ausgeführt und kommuniziert über IPC mit Ihrem übergeordneten Benutzerprozess. Diese Architektur wird gelegentlich auch als Microservice bezeichnet und aus zwei Gründen verwendet:

1) Ihr Benutzerprozess stürzt auch dann nicht ab, wenn die KPL abstürzt

Im Prozess sind möglicherweise Aufgaben enthalten, die nichts mit Kinesis Data Streams zu tun haben. Diese können auch bei einem Absturz der KPL weiterhin durchgeführt werden. Die KPL kann von Ihrem übergeordneten Benutzerprozess auch neu gestartet werden, sodass eine volle Betriebsfähigkeit wiederhergestellt wird (die Funktion ist in den offiziellen Wrappern enthalten).

Ein Beispiel hierfür ist ein Webserver, der Metriken an Kinesis Data Streams sendet. Der Server kann auch noch Seiten bereitstellen, wenn die Komponente Kinesis Data Streams nicht mehr betriebsfähig ist. Ein Absturz des gesamten Servers aufgrund eines Fehlers in der KPL würde einen unnötigen Ausfall verursachen.

2) Es können beliebige Clients unterstützt werden

Es gibt immer Kunden, die Sprachen verwenden, die nicht offiziell unterstützt werden. Diese Kunden sollten die KPL auch problemlos nutzen können.

Matrix zur empfohlenen Verwendung

In der folgenden Verwendungsmatrix sind die empfohlenen Einstellungen für verschiedene Benutzer aufgeführt und Sie werden darüber informiert, ob und wie Sie die KPL verwenden sollten. Beachten Sie, dass bei aktivierter Aggregation auch eine Disaggregation verwendet werden muss, um die Datensätze auf der Konsumentenseite zu extrahieren.

Sprache auf Produzentenseite	Sprache auf Konsumentenseite	KCL-Version	Prüfpunkt-Logik	Können Sie die KPL verwenden?	Einschränkungen
Alles außer Java	*	*	*	Nein	–
Java	Java	Direkte Java-SDK-Verwendung	–	Ja	Bei aktivierter Aggregation muss nach `GetRecords` -Aufrufen die bereitgestellte Disaggregationsbibliothek verwendet werden.
Java	Alles außer Java	Direkte SDK-Verwendung	–	Ja	Aggregation muss deaktiviert werden.
Java	Java	1.3.x	–	Ja	Aggregation muss deaktiviert werden.
Java	Java	1.4.x	Prüfpunkt-Aufruf ohne Argumente	Ja	Keine
Java	Java	1.4.x	Prüfpunkt -Aufruf mit expliziter	Ja	Entweder muss die Aggregation deaktivie

Sprache auf Produzentenseite	Sprache auf Konsumentenseite	KCL-Version	Prüfpunkt-Logik	Können Sie die KPL verwenden?	Einschränkungen
			Sequenznummer		rt oder der Code so geändert werden, dass erweiterte Sequenznummern für das Setzen von Prüfpunkten verwendet werden.
Java	Alles außer Java	1.3.x + mehrsprachiger Daemon + sprachspezifischer Wrapper	–	Ja	Aggregation muss deaktiviert werden.

Schreiben Sie mit der KPL in Ihren Kinesis-Datenstream

In den folgenden Abschnitten wird Beispielcode in einem Schritt vom einfachsten Producer-Code bis hin zum vollständig asynchronen Code dargestellt.

Barebones-Produzentencode

Der folgende Code reicht für die Entwicklung eines einfachen Produzenten aus. Die Benutzerdatensätze der Amazon Kinesis Producer Library (KPL) werden im Hintergrund verarbeitet.

```
// KinesisProducer gets credentials automatically like
// DefaultAWSCredentialsProviderChain.
// It also gets region automatically from the EC2 metadata service.
KinesisProducer kinesis = new KinesisProducer();
```

```
// Put some records
for (int i = 0; i < 100; ++i) {
    ByteBuffer data = ByteBuffer.wrap("myData".getBytes("UTF-8"));
    // doesn't block
    kinesis.addUserRecord("myStream", "myPartitionKey", data);
}
// Do other stuff ...
```

Reagieren Sie synchron auf Ergebnisse

Im vorherigen Beispiel wird vom Code nicht geprüft, ob die KPL-Benutzerdatensätze erfolgreich verarbeitet wurden. Die KPL führt alle Wiederholungen durch, die notwendig sind, um Ausfälle zu kompensieren. Wenn Sie jedoch die Ergebnisse prüfen möchten, können Sie sich diese, wie im folgenden Beispiel gezeigt, mit den `Future`-Objekten ansehen, die von `addUserRecord` zurückgegeben werden (vorheriges Beispiel aus Kontextgründen erneut aufgeführt).

```
KinesisProducer kinesis = new KinesisProducer();

// Put some records and save the Futures
List<Future<UserRecordResult>> putFutures = new
 LinkedList<Future<UserRecordResult>>();
for (int i = 0; i < 100; i++) {
    ByteBuffer data = ByteBuffer.wrap("myData".getBytes("UTF-8"));
    // doesn't block
    putFutures.add(
        kinesis.addUserRecord("myStream", "myPartitionKey", data));
}

// Wait for puts to finish and check the results
for (Future<UserRecordResult> f : putFutures) {
    UserRecordResult result = f.get(); // this does block
    if (result.isSuccessful()) {
        System.out.println("Put record into shard " +
                           result.getShardId());
    } else {
        for (Attempt attempt : result.getAttempts()) {
            // Analyze and respond to the failure
        }
    }
}
```

Reagieren Sie asynchron auf Ergebnisse

Im vorherigen Beispiel wird ein `get()` `Future` Objekt aufgerufen, das die Laufzeit blockiert. Wenn Sie die Laufzeit nicht blockieren möchten, können Sie einen asynchronen Callback verwenden, wie im folgenden Beispiel gezeigt:

```
KinesisProducer kinesis = new KinesisProducer();

FutureCallback<UserRecordResult> myCallback = new FutureCallback<UserRecordResult>() {

    @Override public void onFailure(Throwable t) {
        /* Analyze and respond to the failure  */
    };
    @Override public void onSuccess(UserRecordResult result) {
        /* Respond to the success */
    };
};

for (int i = 0; i < 100; ++i) {
    ByteBuffer data = ByteBuffer.wrap("myData".getBytes("UTF-8"));
    ListenableFuture<UserRecordResult> f = kinesis.addUserRecord("myStream",
 "myPartitionKey", data);
    // If the Future is complete by the time we call addCallback, the callback will be
 invoked immediately.
    Futures.addCallback(f, myCallback);
}
```

Konfiguration der Amazon Kinesis Producer Library

Die Standardeinstellungen reichen für die meisten Anwendungsfälle aus. Bei Bedarf können Sie aber auch das Verhalten von `KinesisProducer` an Ihre Bedürfnisse anpassen. Dazu kann beispielsweise eine Instance der `KinesisProducerConfiguration`-Klasse an den `KinesisProducer`-Konstruktor übergeben werden:

```
KinesisProducerConfiguration config = new KinesisProducerConfiguration()
        .setRecordMaxBufferedTime(3000)
        .setMaxConnections(1)
        .setRequestTimeout(60000)
        .setRegion("us-west-1");

final KinesisProducer kinesisProducer = new KinesisProducer(config);
```

Sie können zudem eine Konfiguration aus einer Eigenschaftendatei laden:

```
KinesisProducerConfiguration config =
 KinesisProducerConfiguration.fromPropertiesFile("default_config.properties");
```

Sie können jeden Pfad und jeden Dateinamen ersetzen, auf den der Benutzerprozess Zugriff hat. Sie können weitere SET-Methoden auf der so erstellten `KinesisProducerConfiguration`-Instance aufrufen, um die Konfiguration anzupassen.

In der Eigenschaftendatei sollten Parameter anhand ihrer Namen in PascalCase angegeben werden. Die Namen stimmen mit denen überein, die in den SET-Methoden der `KinesisProducerConfiguration`-Klasse verwendet werden. Beispiel:

```
RecordMaxBufferedTime = 100
MaxConnections = 4
RequestTimeout = 6000
Region = us-west-1
```

Weitere Informationen zur Verwendung von Konfigurationsparametern und zu Wertgrenzen finden Sie in der Beispieldatei mit den Konfigurationseigenschaften unter GitHub.

Beachten Sie, dass nach dem Initialisieren von `KinesisProducer` eine Änderung der verwendeten `KinesisProducerConfiguration`-Instance keine weiteren Auswirkungen mehr hat. `KinesisProducer` unterstützt derzeit keine dynamische Neukonfiguration.

Implementieren Sie die Deaggregation für Verbraucher

Ab Version 1.4.0 unterstützt die KCL eine automatische Disaggregation von KPL-Benutzerdatensätzen. Code für Konsumentenanwendungen, der mit früheren KCL-Versionen geschrieben wurde, wird nach der Aktualisierung der KCL ohne Änderungen kompiliert. Wenn jedoch eine KPL-Aggregation auf Konsumentenseite verwendet wird, ist beim Checkpointing Folgendes zu beachten: Alle untergeordneten Datensätze innerhalb eines aggregierten Datensatzes haben dieselbe Sequenznummer. Somit müssen zusätzliche Daten mit dem Prüfpunkt gespeichert werden, wenn Sie zwischen untergeordneten Datensätze unterscheiden müssen. Diese zusätzlichen Daten werden als Teilsequenznummer bezeichnet.

Optionen

- Migrieren Sie von früheren Versionen der KCL

- Verwenden Sie KCL-Erweiterungen für die KPL-Deaggregation
- Direkt verwenden GetRecords

Migrieren Sie von früheren Versionen der KCL

Sie müssen Ihre vorhandenen Aufrufe nicht ändern, um Checkpointing mit Aggregation durchzuführen. Sie können nach wie vor alle Datensätze erfolgreich abrufen, die in Kinesis Data Streams gespeichert sind. Die KCL bietet jetzt zwei neue Checkpoint-Operationen zur Unterstützung bestimmter Anwendungsfälle, die im Folgenden beschrieben werden.

Wenn Ihr vorhandener Code vor der KPL-Unterstützung für die KCL geschrieben wurde und Ihre Checkpoint-Operation ohne Argumente aufgerufen wird, entspricht dies dem Checkpointing der Sequenznummer des letzten KPL-Benutzerdatensatzes im Batch. Wird Ihre Prüfpunkt-Operation mit einer Sequenznummernzeichenfolge aufgerufen, entspricht dies dem Checkpointing der angegebenen Sequenznummer des Stapels zusammen mit der impliziten Teilsequenznummer 0 (null).

Das Aufrufen der neuen KCL-Prüfpunkt-Operation `checkpoint()` ohne Argumente entspricht semantisch dem Checkpointing der Sequenznummer des letzten `Record`-Aufrufs im Stapels zusammen mit der impliziten Teilsequenznummer 0 (null).

Das Aufrufen der neuen KCL-Prüfpunkt-Operation `checkpoint(Record record)` entspricht semantisch dem Checkpointing der angegebenen Sequenznummer von `Record` zusammen mit der impliziten Teilsequenznummer 0 (null). Handelt es sich beim `Record`-Aufruf um einen `UserRecord`, wird ein Checkpointing der Sequenznummer und Teilsequenznummer von `UserRecord` durchgeführt.

Das Aufrufen der neuen KCL-Prüfpunkt-Operation `checkpoint(String sequenceNumber, long subSequenceNumber)` führt explizit zu einem Checkpointing der angegebenen Sequenznummer zusammen mit der angegebenen Teilsequenznummer.

In all diesen Fällen kann, nachdem der Prüfpunkt in der Amazon-DynamoDB-Prüfpunkttabelle gespeichert wurde, die KCL das Abrufen der Datensätze fehlerfrei wieder aufnehmen, auch wenn die Anwendung abstürzt und neu gestartet wird. Sind mehr Datensätze in der Sequenz enthalten, beginnt das Abrufen mit dem Datensatz, dem die nächste Teilsequenznummer zugeordnet wurde, innerhalb des Datensatzes mit der Sequenznummer, für die zuletzt ein Prüfpunkt gesetzt wurde. Enthält der letzte Prüfpunkt die allerletzte Teilsequenznummer des vorherigen Sequenznummerndatensatzes, beginnt das Abrufen mit dem Datensatz, dem die nächst folgende Sequenznummer zugeordnet ist.

Im nächsten Abschnitt werden Einzelheiten zur Sequenz- und Folgesequenzprüfung für Benutzer beschrieben, die vermeiden müssen, dass Datensätze übersprungen und dupliziert werden. Wenn das Überspringen (oder Duplizieren) bei einem Stopp und Neustart der Datensatzverarbeitung Ihres Konsumenten keine Rolle spielt, können Sie Ihren vorhandenen Code ohne Änderungen ausführen.

Verwenden Sie KCL-Erweiterungen für die KPL-Deaggregation

Die KPL-Deaggregation kann Teilsequenz-Checkpoints beinhalten. Zur Unterstützung des Checkpointings einer Teilsequenz wurde eine `UserRecord`-Klasse zur KCL hinzugefügt:

```
public class UserRecord extends Record {
    public long getSubSequenceNumber() {
    /* ... */
    }
    @Override
    public int hashCode() {
    /* contract-satisfying implementation */
    }
    @Override
    public boolean equals(Object obj) {
    /* contract-satisfying implementation */
    }
}
```

Diese Klasse wird nun anstelle von `Record` verwendet. Sie führt nicht zu Fehlern im vorhandenen Code, da es sich um eine Subklasse von `Record` handelt. Die `UserRecord`-Klasse repräsentiert sowohl tatsächlich untergeordnete Datensätze als auch standardmäßige, nicht aggregierte Datensätze. Nicht-aggregierte Datensätze sind aggregierte Datensätze mit genau einem untergeordneten Datensatz.

Darüber hinaus wurden zwei neue Operationen zu `IRecordProcessorCheckpointer` hinzugefügt:

```
public void checkpoint(Record record);
public void checkpoint(String sequenceNumber, long subSequenceNumber);
```

Führen Sie die folgende Konvertierung durch, um mit dem Checkpointing einer Teilsequenznummer zu beginnen: Ändern Sie folgenden Formularcode:

```
checkpointer.checkpoint(record.getSequenceNumber());
```

Neue Formularcode:

```
checkpointer.checkpoint(record);
```

Wir empfehlen für das Checkpointing der Teilsequenz das `checkpoint(Record record)`-Formular. Wenn Sie jedoch bereits `sequenceNumbers` in Zeichenfolgen für das Checkpointing gespeichert haben, sollten Sie nun auch `subSequenceNumber` speichern, wie im folgenden Beispiel gezeigt:

```
String sequenceNumber = record.getSequenceNumber();
long subSequenceNumber = ((UserRecord) record).getSubSequenceNumber();  // ... do other
 processing
checkpointer.checkpoint(sequenceNumber, subSequenceNumber);
```

Die Umwandlung von nach ist `UserRecord` immer erfolgreich`Record`, da die Implementierung immer verwendet. `UserRecord` Wenn Sie keine arithmetischen Operationen für die Sequenznummern durchführen müssen, ist dieser Ansatz nicht zu empfehlen.

Während der Verarbeitung der KPL-Benutzerdatensätze schreibt die KCL die Teilsequenznummer für jede Zeile als zusätzliches Feld in die Amazon DynamoDB. Frühere Versionen der KCL nutzten `AFTER_SEQUENCE_NUMBER` zum Abrufen von Datensätzen bei der Wiederaufnahme von Prüfpunkten. Die aktuelle KCL mit KPL-Support verwendet stattdessen `AT_SEQUENCE_NUMBER`. Wenn der Datensatz bei der Sequenznummer abgerufen wird, bei der ein Prüfpunkt gesetzt wurde, wird die Teilsequenznummer, für die ein Checkpointing durchgeführt wurde, geprüft und untergeordnete Datensätze gegebenenfalls ausgelassen (möglicherweise alle, wenn beim letzten Datensatz ein Prüfpunkt gesetzt wurde). Nochmals: Nicht aggregierte Datensätze können als aggregierte Datensätze mit einem einzelnen untergeordneten Datensatz betrachtet werden, sodass derselbe Algorithmus sowohl für aggregierte als auch für nicht aggregierte Datensätze funktioniert.

Direkt verwenden GetRecords

Sie können sich auch gegen die Verwendung der KCL entscheiden und stattdessen direkt die API-Operation `GetRecords` aufrufen, um Datensätze aus Kinesis Data Streams abzurufen. Zum Entpacken dieser abgerufenen Datensätze in Ihre ursprünglichen KPL-Benutzerdatensätze rufen Sie eine der folgenden statischen Operationen in `UserRecord.java` auf:

```
public static List<Record> deaggregate(List<Record> records)
```

```
public static List<UserRecord> deaggregate(List<UserRecord> records, BigInteger
 startingHashKey, BigInteger endingHashKey)
```

Die erste Operation verwendet den Standardwert `0` (null) für `startingHashKey` und den Standardwert `2^128 -1` für `endingHashKey`.

Jede dieser Operationen führt eine Disaggregation der vorhandenen Liste der Datensätze aus Kinesis Data Streams in eine Liste mit KPL-Benutzerdatensätzen durch. Alle KPL-Benutzerdatensätze, deren expliziter Hash- oder Partitionsschlüssel außerhalb des Bereichs von `startingHashKey` (einschließlich) liegt, sowie `endingHashKey` (einschließlich) werden aus der zurückgegebenen Datensatzliste entfernt.

Verwenden Sie die KPL mit Amazon Data Firehose

Wenn Sie die Kinesis Producer Library (KPL) verwenden, um Daten an einen Kinesis Data Stream zu schreiben, können Sie die Aggregation dazu verwenden, die an diesen Kinesis Data Stream geschriebenen Datensätze zu kombinieren. Wenn Sie diesen Datenstrom dann als Quelle für Ihren Firehose-Lieferstream verwenden, deaggregiert Firehose die Datensätze, bevor sie an das Ziel geliefert werden. Wenn Sie Ihren Delivery Stream so konfigurieren, dass er die Daten transformiert, deaggregiert Firehose die Datensätze, bevor es sie zustellt. AWS Lambda Weitere Informationen finden Sie unter Schreiben zu Amazon Firehose mithilfe von Kinesis Data Streams.

Verwenden Sie die KPL mit der Schema Registry AWS Glue

Sie können Ihre Kinesis-Datenströme in die AWS Glue Schema Registry integrieren. Mit der AWS Glue Schema Registry können Sie Schemas zentral erkennen, steuern und weiterentwickeln und gleichzeitig sicherstellen, dass die erstellten Daten kontinuierlich anhand eines registrierten Schemas validiert werden. Ein Schema definiert die Struktur und das Format eines Datensatzes. Ein Schema ist eine versionierte Spezifikation für zuverlässige Datenveröffentlichung, -nutzung oder -speicherung. Mit der AWS Glue Schema Registry können Sie die end-to-end Datenqualität und Datenverwaltung in Ihren Streaming-Anwendungen verbessern. Weitere Informationen finden Sie unter AWS Glue Schema Registry. Alternativ können Sie diese Integration über die Bibliotheken KPL und Kinesis Client Library (KCL) in Java einrichten.

> **Important**
>
> Derzeit wird die Integration von Kinesis Data Streams und AWS Glue Schema-Registry nur für Kinesis-Datenstreams unterstützt, die in Java implementierte KPL-Producer verwenden. Mehrsprachige Unterstützung wird nicht bereitgestellt.

Detaillierte Anweisungen zur Einrichtung der Integration von Kinesis Data Streams mit Schema Registry mithilfe der KPL finden Sie im Abschnitt „Interaktion mit Daten mithilfe der KPL/KCL Bibliotheken“ unter Anwendungsfall: Integration von Amazon Kinesis Data Streams mit der AWS Glue Schema Registry.

Konfigurieren Sie die KPL-Proxykonfiguration

Für Anwendungen, die keine direkte Verbindung zum Internet herstellen können, unterstützen alle AWS SDK-Clients die Verwendung von HTTP- oder HTTPS-Proxys. In einer typischen Unternehmensumgebung muss der gesamte ausgehende Netzwerkverkehr über Proxyserver geleitet werden. Wenn Ihre Anwendung Kinesis Producer Library (KPL) zum Sammeln und Senden von Daten AWS in eine Umgebung verwendet, die Proxyserver verwendet, benötigt Ihre Anwendung eine KPL-Proxykonfiguration. KPL ist eine High-Level-Bibliothek, die auf dem AWS Kinesis SDK aufbaut. Sie ist in einen systemeigenen Prozess und einen Wrapper aufgeteilt. Der native Prozess führt alle Aufgaben der Verarbeitung und des Sendens von Datensätzen aus, während der Wrapper den nativen Prozess verwaltet und mit ihm kommuniziert. Weitere Informationen finden Sie unter Implementieren effizienter und zuverlässiger Produzenten mit der Amazon Kinesis Producer Library.

Der Wrapper ist in Java geschrieben und der native Prozess ist in C++ unter Verwendung des Kinesis SDK geschrieben. KPL-Version 0.14.7 und höher unterstützt jetzt die Proxykonfiguration im Java-Wrapper, der alle Proxykonfigurationen an den nativen Prozess übergeben kann. Weitere Informationen finden Sie unter https://github.com/awslabs/amazon-kinesis-producer/releases/tag/v0.14.7.

Alternativ können Sie den folgenden Code verwenden, um Proxykonfigurationen zu Ihren KPL-Anwendungen hinzuzufügen.

```
KinesisProducerConfiguration configuration = new KinesisProducerConfiguration();
// Next 4 lines used to configure proxy
configuration.setProxyHost("10.0.0.0"); // required
```

```
configuration.setProxyPort(3128); // default port is set to 443
configuration.setProxyUserName("username"); // no default
configuration.setProxyPassword("password"); // no default

KinesisProducer kinesisProducer = new KinesisProducer(configuration);
```

Richtlinie zum Lebenszyklus der KPL-Version

In diesem Thema wird die Versions-Lebenszyklus-Richtlinie für die Amazon Kinesis Producer Library (KPL) beschrieben. AWS stellt regelmäßig neue Versionen für KPL-Versionen bereit, um neue Funktionen und Verbesserungen, Bugfixes, Sicherheitspatches und Abhängigkeitsupdates zu unterstützen. Wir empfehlen Ihnen, up-to-date bei KPL-Versionen zu bleiben, um über die neuesten Funktionen, Sicherheitsupdates und die zugrunde liegenden Abhängigkeiten auf dem Laufenden zu bleiben. Wir empfehlen nicht, weiterhin eine KPL-Version zu verwenden, die nicht unterstützt wird.

Der Lebenszyklus wichtiger KPL-Versionen besteht aus den folgenden drei Phasen:

- Allgemeine Verfügbarkeit (GA) — Während dieser Phase wird die Hauptversion vollständig unterstützt. AWS bietet regelmäßige Versionen von Neben- und Patch-Versionen, die Unterstützung für neue Funktionen oder API-Updates für Kinesis Data Streams sowie Fehler- und Sicherheitskorrekturen beinhalten.
- Wartungsmodus — AWS Beschränkt die Veröffentlichung von Patch-Versionen auf kritische Bugfixes und Sicherheitsprobleme. Die Hauptversion erhält keine Updates für neue Funktionen oder APIs für Kinesis Data Streams.
- E nd-of-support — Die Hauptversion wird keine Updates oder Releases mehr erhalten. Zuvor veröffentlichte Versionen werden weiterhin über öffentliche Paketmanager verfügbar sein und der Code bleibt aktiviert GitHub. Die Verwendung einer Version, die erreicht wurde, end-of-support erfolgt nach eigenem Ermessen des Benutzers. Wir empfehlen Ihnen, auf die neueste Hauptversion zu aktualisieren.

Hauptversion	Aktuelle Phase	Datum der Veröffentlichung	Datum des Wartungsmodus	End-of-support Datum
KPL 0.x	Wartungsmodus	2015-06-02	17.04.2025	2026-01-30

Hauptversion	Aktuelle Phase	Datum der Veröffentlichung	Datum des Wartungsmodus	End-of-support Datum
KPL 1.x	Allgemeine Verfügbarkeit	2024-12-15	--	--

Entwickeln Sie Produzenten mithilfe der Amazon Kinesis Data Streams Streams-API mit dem AWS SDK für Java

Sie können Producer mithilfe der Amazon Kinesis Data Streams Streams-API mit dem AWS SDK for Java entwickeln. Wenn Sie Kinesis Data Streams zum ersten Mal verwenden, sollten Sie sich zunächst mit den Konzepten und der Terminologie in Was ist Amazon Kinesis Data Streams? und Verwenden Sie die AWS CLI , um Amazon Kinesis Data Streams Streams-Operationen durchzuführen vertraut machen.

In den Beispielen wird die API für Kinesis Data Streams behandelt und das AWS -SDK for Java eingesetzt, um Daten zu einem Stream hinzuzufügen. Für die meisten Anwendungsfälle sollten Sie jedoch die KPL-Bibliothek von Kinesis Data Streams verwenden. Weitere Informationen finden Sie unter Entwickeln Sie Produzenten mithilfe der Amazon Kinesis Producer Library (KPL).

Anhand des Java-Beispiel-Codes in diesem Abschnitt, der nach Operationstyp logisch unterteilt ist, wird gezeigt, wie Sie grundlegende API-Vorgänge mit Kinesis Data Streams durchführen. Diese Beispiele stellen keinen produktionsbereiten Code dar, d. h. es werden nicht alle möglichen Ausnahmen geprüft und es werden nicht alle möglichen Sicherheits- oder Leistungsüberlegungen berücksichtigt. Sie können zudem die API für Kinesis Data Streams mit anderen Programmiersprachen aufrufen. Weitere Informationen zu allen verfügbaren AWS SDKs Produkten finden Sie unter Start Developing with Amazon Web Services.

Für jede Aufgabe gibt es Voraussetzungen. So können Sie beispielsweise erst dann Daten zu einem Stream hinzufügen, wenn Sie einen erstellt haben. Deshalb müssen Sie einen Client anlegen. Weitere Informationen finden Sie unter Kinesis-Datenstreams erstellen und verwalten.

Themen

- Daten zu einem Stream hinzufügen
- Interagieren Sie mit Daten mithilfe der AWS Glue Schema Registry

Daten zu einem Stream hinzufügen

Sobald ein Stream eingerichtet wurde, können Sie Daten in Form von Datensätzen hinzufügen. Ein Datensatz ist eine Datenstruktur, die die zu verarbeitenden Daten in Form eines Daten-Blobs enthält. Nach dem Speichern der Daten im Datensatz prüft, interpretiert und ändert Kinesis Data Streams die Daten nicht. Jeder Datensatz verfügt zudem über eine zugeordnete Sequenznummer und einen Partitionsschlüssel.

Die API für Kinesis Data Streams unterstützt das Hinzufügen von Daten zu einem Stream mittels zweier Operationen: `PutRecords` und `PutRecord`. Die Operation `PutRecords` sendet per HTTP-Anforderung mehrere Datensätze an Ihren Stream und die Operation `PutRecord` sendet immer nur einen Datensatz, dabei ist für jeden Datensatz eine separate HTTP-Anforderung erforderlich. Für die meisten Anwendungen sollten Sie `PutRecords` bevorzugen, da diese Operation pro Datenproduzent einen höheren Durchsatz erzielt. Weitere Informationen zu diesen Operationen finden Sie unten in separaten Unterabschnitten.

Themen

- Fügen Sie mehrere Datensätze hinzu mit PutRecords
- Fügen Sie einen einzelnen Datensatz hinzu mit PutRecord

Beachten Sie Folgendes: Wenn Ihre Quellanwendung Daten mit der API für Kinesis Data Streams zum Stream hinzufügt, gibt es vermutlich eine oder mehrere Konsumentenanwendungen, die gleichzeitig Daten aus dem Stream verarbeiten. Weitere Informationen dazu, wie Konsumenten Daten über die API für Kinesis Data Streams abrufen, erhalten Sie unter Daten aus einem Stream abrufen.

> **Important**
>
> Ändern Sie den Aufbewahrungszeitraum für Daten

Fügen Sie mehrere Datensätze hinzu mit PutRecords

Die `PutRecords`-Operation sendet in einer einzelnen Anforderung mehrere Datensätze an Kinesis Data Streams. Mit `PutRecords` erzielen Produzenten einen höheren Durchsatz, wenn sie Daten an ihren Kinesis-Datenstrom senden. Jede `PutRecords`-Anfrage kann bis zu 500 Datensätze unterstützen. Die maximale Größe jedes Datensatzes in der Anforderung beträgt 1 MB bis zu einem Limit von 5 MB für die gesamte Anforderung einschließlich Partitionsschlüsseln. Wie bei der

unten beschriebenen einzelnen `PutRecord`-Operation nutzt `PutRecords` Sequenznummern und Partitionsschlüssel. Der `PutRecord`-Parameter `SequenceNumberForOrdering` ist jedoch nicht in einem `PutRecords`-Aufruf enthalten. Die `PutRecords`-Operation versucht, alle Datensätze in der natürlichen Reihenfolge der Anforderung zu verarbeiten.

Jeder Datensatz hat eine eindeutige Sequenznummer. Die Sequenznummer wird von Kinesis Data Streams zugewiesen, nachdem Sie `client.putRecords` aufgerufen haben, um die Datensätze zum Stream hinzuzufügen. Sequenznummern für denselben Partitionsschlüssel steigen in der Regel im Laufe der Zeit; je länger die Zeitdauer zwischen `PutRecords`-Anforderungen ist, desto größer wird die Sequenznummer.

> **Note**
>
> Sequenznummern können nicht als Index für Datensätze innerhalb desselben Streams verwendet werden. Nutzen Sie für die logische Unterteilung von Datensätzen Partitionsschlüssel oder erstellen Sie für jeden Datensatz einen separaten Stream.

Eine `PutRecords`-Anforderung kann Datensätze mit unterschiedlichen Partitionsschlüsseln enthalten. Der Anforderung gilt für einen Stream. Jede Anforderung kann eine beliebige Kombination aus Partitionsschlüsseln und Datenschlüsseln enthalten, sofern die Grenzwerte der Anforderung nicht überschritten werden. Anforderungen mit vielen verschiedenen Partitionsschlüsseln an Streams mit vielen verschiedenen Shards sind im Allgemeinen schneller als Anforderungen mit wenigen Partitionsschlüsseln an Streams mit nur wenigen Shards. Die Anzahl der Partitionsschlüssel sollte viel größer sein als die Anzahl der Shards, um Latenzzeiten zu verkürzen und einen maximalen Durchsatz zu erzielen.

PutRecords Beispiel

Der folgende Code erstellt 100 Datensätze mit sequenziellen Partitionsschlüsseln und übergibt diese an einen Stream namens `DataStream`.

```
        AmazonKinesisClientBuilder clientBuilder =
AmazonKinesisClientBuilder.standard();

        clientBuilder.setRegion(regionName);
        clientBuilder.setCredentials(credentialsProvider);
        clientBuilder.setClientConfiguration(config);

        AmazonKinesis kinesisClient = clientBuilder.build();
```

```
        PutRecordsRequest putRecordsRequest  = new PutRecordsRequest();
        putRecordsRequest.setStreamName(streamName);
        List <PutRecordsRequestEntry> putRecordsRequestEntryList  = new ArrayList<>();
        for (int i = 0; i < 100; i++) {
            PutRecordsRequestEntry putRecordsRequestEntry  = new
PutRecordsRequestEntry();

putRecordsRequestEntry.setData(ByteBuffer.wrap(String.valueOf(i).getBytes()));
            putRecordsRequestEntry.setPartitionKey(String.format("partitionKey-%d",
i));
            putRecordsRequestEntryList.add(putRecordsRequestEntry);
        }

        putRecordsRequest.setRecords(putRecordsRequestEntryList);
        PutRecordsResult putRecordsResult  =
kinesisClient.putRecords(putRecordsRequest);
        System.out.println("Put Result" + putRecordsResult);
```

Die `PutRecords`-Antwort umfasst ein Array von `Records`-Antworten. Jeder Eintrag im Antwort-Array korreliert direkt mit einem Eintrag im Anforderungs-Array und zwar in natürlicher Reihenfolge von oben nach unten wie in der Anforderung und der Antwort. Das `Records`-Antwort-Array enthält stets die gleiche Anzahl an Datensätzen wie das Anforderungs-Array.

Behandeln Sie Fehler bei der Verwendung PutRecords

Standardmäßig führt ein Fehler bei einzelnen Datensätzen innerhalb einer Anforderung nicht zur Beendigung der Verarbeitung nachfolgender Datensätze in einer `PutRecords`-Anforderung. Das bedeutet, dass ein `Records`-Antwort-Array sowohl erfolgreich verarbeitete als auch nicht erfolgreich verarbeitete Datensätze enthält. Sie müssen nicht erfolgreich verarbeitete Datensätze erkennen und in einen nachfolgenden Aufruf aufnehmen.

Erfolgreich verarbeitete Datensätze enthalten `SequenceNumber`- und `ShardID`-Werte, nicht erfolgreiche verarbeitete Datensätze enthalten `ErrorCode`-und `ErrorMessage`-Werte. Der `ErrorCode`-Parameter gibt den Fehlertyp an. Folgende Werte sind möglich: `ProvisionedThroughputExceededException` oder `InternalFailure`. `ErrorMessage` enthält weitere Informationen zur `ProvisionedThroughputExceededException`-Ausnahme, einschließlich Konto-ID, Stream-Name und Shard-ID des gedrosselten Datensatzes. Im folgenden Beispiel umfasst eine `PutRecords`-Anforderung drei Datensätze. Der zweite Datensatz schlägt fehl und wird in der Antwort angegeben.

Example PutRecords Syntax anfordern

```
{
    "Records": [
        {
     "Data": "XzxkYXRhPl8w",
     "PartitionKey": "partitionKey1"
        },
        {
     "Data": "AbceddeRFfg12asd",
     "PartitionKey": "partitionKey1"
        },
        {
     "Data": "KFpcd98*7nd1",
     "PartitionKey": "partitionKey3"
        }
    ],
    "StreamName": "myStream"
}
```

Example PutRecords Syntax der Antwort

```
{
    "FailedRecordCount”: 1,
    "Records": [
        {
     "SequenceNumber": "21269319989900637946712965403778482371",
     "ShardId": "shardId-000000000001"

        },
        {
     “ErrorCode":”ProvisionedThroughputExceededException”,
     “ErrorMessage": "Rate exceeded for shard shardId-000000000001 in stream
 exampleStreamName under account 111111111111."

        },
        {
     "SequenceNumber": "21269319989999637946712965403778482985",
     "ShardId": "shardId-000000000002"
        }
    ]
}
```

Datensätze, die nicht erfolgreich verarbeitet wurden, können in nachfolgende `PutRecords`-Anforderungen aufgenommen werden. Überprüfen Sie zuerst den Parameter `FailedRecordCount` im `putRecordsResult`, um zu bestätigen, ob Datensätze in der Anforderung fehlgeschlagen sind. Wenn dies der Fall ist, darf jeder `putRecordsEntry`, dessen `ErrorCode` nicht `null` ist, nicht in der nächsten Anforderung hinzugefügt werden. Ein Beispiel für diese Art von Handler finden Sie in folgendem Code:

Example PutRecords Fehlerhandler

```
PutRecordsRequest putRecordsRequest = new PutRecordsRequest();
putRecordsRequest.setStreamName(myStreamName);
List<PutRecordsRequestEntry> putRecordsRequestEntryList = new ArrayList<>();
for (int j = 0; j < 100; j++) {
    PutRecordsRequestEntry putRecordsRequestEntry = new PutRecordsRequestEntry();
    putRecordsRequestEntry.setData(ByteBuffer.wrap(String.valueOf(j).getBytes()));
    putRecordsRequestEntry.setPartitionKey(String.format("partitionKey-%d", j));
    putRecordsRequestEntryList.add(putRecordsRequestEntry);
}

putRecordsRequest.setRecords(putRecordsRequestEntryList);
PutRecordsResult putRecordsResult = amazonKinesisClient.putRecords(putRecordsRequest);

while (putRecordsResult.getFailedRecordCount() > 0) {
    final List<PutRecordsRequestEntry> failedRecordsList = new ArrayList<>();
    final List<PutRecordsResultEntry> putRecordsResultEntryList =
 putRecordsResult.getRecords();
    for (int i = 0; i < putRecordsResultEntryList.size(); i++) {
        final PutRecordsRequestEntry putRecordRequestEntry =
 putRecordsRequestEntryList.get(i);
        final PutRecordsResultEntry putRecordsResultEntry =
 putRecordsResultEntryList.get(i);
        if (putRecordsResultEntry.getErrorCode() != null) {
            failedRecordsList.add(putRecordRequestEntry);
        }
    }
    putRecordsRequestEntryList = failedRecordsList;
    putRecordsRequest.setRecords(putRecordsRequestEntryList);
    putRecordsResult = amazonKinesisClient.putRecords(putRecordsRequest);
}
```

Fügen Sie einen einzelnen Datensatz hinzu mit PutRecord

Jeder Aufruf von `PutRecord` wird auf einem einzelnen Datensatz ausgeführt. Bevorzugen Sie die `PutRecords`-Operation, die in Fügen Sie mehrere Datensätze hinzu mit PutRecords beschrieben ist, es sei denn, Ihre Anwendung muss stets einzelne Datensätze per Anforderung senden oder es liegen andere Gründe vor, die gegen den Einsatz von `PutRecords` sprechen.

Jeder Datensatz hat eine eindeutige Sequenznummer. Die Sequenznummer wird von Kinesis Data Streams zugewiesen, nachdem Sie `client.putRecord` aufgerufen haben, um den Datensatz zum Stream hinzuzufügen. Sequenznummern für denselben Partitionsschlüssel steigen in der Regel im Laufe der Zeit; je länger die Zeitdauer zwischen `PutRecord`-Anforderungen ist, desto größer wird die Sequenznummer.

Wenn nacheinander viele PUT-Operationen durchgeführt werden, steigen die zurückgegebenen Sequenznummern nicht zwangsläufig an, da Kinesis Data Streams diese als gleichzeitig interpretiert. Um steigende Sequenznummern für denselben Partitionsschlüssel sicherzustellen, sollten Sie den `SequenceNumberForOrdering`-Parameter so verwenden, wie es im PutRecord Beispiel-Codebeispiel gezeigt wird.

Unabhängig davon, ob Sie `SequenceNumberForOrdering` verwenden, werden Datensätze, die Kinesis Data Streams über einen `GetRecords`-Aufruf empfängt, streng nach Sequenznummern sortiert.

> **Note**
>
> Sequenznummern können nicht als Index für Datensätze innerhalb desselben Streams verwendet werden. Nutzen Sie für die logische Unterteilung von Datensätzen Partitionsschlüssel oder erstellen Sie für jeden Datensatz einen separaten Stream.

Ein Partitionsschlüssel wird verwendet, um Daten in einem Stream zu gruppieren. Ein Datensatz wird einem Shard innerhalb des Streams basierend auf dessen Partitionsschlüssel zugewiesen. Insbesondere Kinesis Data Streams verwendet den Partitionsschlüssel als Eingabe für eine Hash-Funktion, die den Partitionsschlüssel (und die zugehörigen Daten) einem bestimmten Shard zuordnet.

Aufgrund dieses Hashing-Mechanismus werden alle Datensätze mit demselben Partitionsschlüssel zum selben Shard im Stream zugeordnet. Wenn jedoch die Anzahl der Partitionsschlüssel die Anzahl der Shards übersteigt, enthalten einige Shards zwangsläufig Datensätze mit unterschiedlichen Partitionsschlüsseln. Vom Design-Standpunkt aus sollte die Anzahl der Shards (die über die

setShardCount-Methode von CreateStreamRequest angegeben wird) wesentlich geringer sein als die Anzahl der eindeutigen Partitionsschlüssel und das Datenvolumen für einen einzelnen Partitionsschlüssel sollte wesentlich geringer sein, als die Shard-Kapazität, um sicherzustellen, dass alle Shards gut ausgelastet sind.

PutRecord Beispiel

Der folgende Code erstellt 10 Datensätze, die auf zwei Partitionsschlüssel verteilt werden, und fügt diese einem Stream namens myStreamName hinzu.

```
for (int j = 0; j < 10; j++)
{
  PutRecordRequest putRecordRequest = new PutRecordRequest();
  putRecordRequest.setStreamName( myStreamName );
  putRecordRequest.setData(ByteBuffer.wrap( String.format( "testData-%d",
 j ).getBytes() ));
  putRecordRequest.setPartitionKey( String.format( "partitionKey-%d", j/5 ));
  putRecordRequest.setSequenceNumberForOrdering( sequenceNumberOfPreviousRecord );
  PutRecordResult putRecordResult = client.putRecord( putRecordRequest );
  sequenceNumberOfPreviousRecord = putRecordResult.getSequenceNumber();
}
```

Beim zuvor beschriebenen Codebeispiel wird setSequenceNumberForOrdering eingesetzt, um eine aufsteigende Reihenfolge innerhalb der einzelnen Partitionsschlüssel zu gewährleisten. Für eine effektive Verwendung des Parameters setzen Sie die SequenceNumberForOrdering des aktuellen Datensatzes (Datensatz n) auf die Sequenznummer des vorhergehenden Datensatzes (Datensatz n-1). Rufen Sie getSequenceNumber auf dem Ergebnis von putRecord auf, um die Sequenznummer des Datensatzes zu erhalten, der zum Stream hinzugefügt wurde.

Der SequenceNumberForOrdering-Parameter stellt strikt aufsteigende Sequenznummern für denselben Partitionsschlüssel sicher. SequenceNumberForOrdering stellt keine Datensatzsortierung bei mehreren Partitionsschlüsseln bereit.

Interagieren Sie mit Daten mithilfe der AWS Glue Schema Registry

Sie können Ihre Kinesis-Datenströme in die AWS Glue Schema Registry integrieren. Mit der AWS Glue Schema Registry können Sie Schemas zentral erkennen, steuern und weiterentwickeln und gleichzeitig sicherstellen, dass die erstellten Daten kontinuierlich anhand eines registrierten Schemas validiert werden. Ein Schema definiert die Struktur und das Format eines Datensatzes. Ein Schema ist eine versionierte Spezifikation für zuverlässige Datenveröffentlichung, -nutzung oder

-speicherung. Mit der AWS Glue Schema Registry können Sie die end-to-end Datenqualität und die Datenverwaltung in Ihren Streaming-Anwendungen verbessern. Weitere Informationen finden Sie unter AWS Glue Schema Registry. Eine Möglichkeit, diese Integration einzurichten, sind die `PutRecords` im AWS Java-SDK APIs verfügbaren `PutRecord` Kinesis Data Streams.

Detaillierte Anweisungen zur Einrichtung der Integration von Kinesis Data Streams mit der Schemaregistry mithilfe von PutRecords und PutRecord Kinesis Data Streams APIs finden Sie im Abschnitt „Interaktion mit Daten mithilfe der Kinesis Data Streams APIs“ unter Anwendungsfall: Integration von Amazon Kinesis Data Streams mit der AWS Glue Schema Registry.

Schreiben Sie mit Kinesis Agent in Amazon Kinesis Data Streams

Kinesis Agent ist eine eigenständige Java-Anwendung, mit der Sie Daten einfach erfassen und an Kinesis Data Streams senden können. Der Agent überwacht kontinuierlich eine Reihe von Dateien und sendet neue Daten an Ihren Stream. Der Agent übernimmt die Dateirotation, das Checkpointing und die Wiederholung bei Fehlern. Er sorgt für eine zuverlässige, zeitgerechte und einfache Bereitstellung Ihrer Daten. Außerdem werden CloudWatch Amazon-Metriken ausgegeben, damit Sie den Streaming-Prozess besser überwachen und Fehler beheben können.

Standardmäßig werden Datensätze aus den einzelnen Dateien anhand des Zeilenumbruchzeichens (`'\n'`) analysiert. Der Agent kann jedoch auch für die Analyse mehrzeiliger Datensätze konfiguriert werden (siehe Geben Sie die Einstellungen für die Agentenkonfiguration an).

Sie können den Agenten in Linux-Serverumgebungen installieren, beispielsweise auf Webservern, Protokollservern und Datenbankservern. Nach der Installation des Agenten konfigurieren Sie diesen, indem Sie die zu überwachenden Dateien sowie den Stream für die Daten angeben. Nachdem der Agent konfiguriert wurde, sammelt er permanent Daten aus den Dateien und sendet sie zuverlässig an den Stream.

Themen

- Erfüllen Sie die Voraussetzungen für Kinesis Agent
- Herunterladen und Installieren des Agenten
- Konfigurieren und starten Sie den Agenten
- Geben Sie die Einstellungen für die Agentenkonfiguration an
- Überwachen Sie mehrere Dateiverzeichnisse und schreiben Sie in mehrere Streams
- Verwenden Sie den Agenten zur Vorverarbeitung von Daten
- Verwenden Sie CLI-Befehle für Agenten

- Häufig gestellte Fragen

Erfüllen Sie die Voraussetzungen für Kinesis Agent

- Sie benötigen entweder ein Amazon Linux AMI-Betriebssystem, Version 2015.09 oder höher, oder Version 7 oder höher von Red Hat Enterprise Linux.
- Starten Sie Ihre EC2 Instance, wenn Sie Amazon EC2 verwenden, um den Agenten auszuführen.
- Verwalten Sie Ihre AWS Anmeldeinformationen mit einer der folgenden Methoden:
  - Geben Sie eine IAM-Rolle an, wenn Sie Ihre EC2 Instance starten.
  - Geben Sie die AWS Anmeldeinformationen an, wenn Sie den Agenten konfigurieren (siehe awsAccessKeyID und awsSecretAccessSchlüssel).
  - Bearbeiten Sie`/etc/sysconfig/aws-kinesis-agent`, um Ihre Region und Ihre AWS Zugriffsschlüssel anzugeben.
  - Wenn sich Ihre EC2-Instance in einem anderen AWS Konto befindet, erstellen Sie eine IAM-Rolle, um Zugriff auf den Kinesis Data Streams Streams-Dienst zu gewähren, und geben Sie diese Rolle bei der Konfiguration des Agenten an (siehe assumeRoleARN und Id). assumeRoleExternal Verwenden Sie eine der vorherigen Methoden, um die AWS Anmeldeinformationen eines Benutzers im anderen Konto anzugeben, der berechtigt ist, diese Rolle anzunehmen.
- Die von Ihnen angegebene IAM-Rolle oder die AWS Anmeldeinformationen müssen berechtigt sein, den Kinesis Data Streams PutRecordsStreams-Vorgang auszuführen, damit der Agent Daten an Ihren Stream senden kann. Wenn Sie die CloudWatch Überwachung für den Agenten aktivieren, ist auch eine Genehmigung zur Durchführung des CloudWatch PutMetricDataVorgangs erforderlich. Weitere Informationen finden Sie unter Steuern des Zugriffs auf Amazon Kinesis Data Streams Streams-Ressourcen mithilfe von IAMÜberwachen Sie den Zustand des Kinesis Data Streams Streams-Agenten mit Amazon CloudWatch, und CloudWatch Zugriffskontrolle.

Herunterladen und Installieren des Agenten

Stellen Sie zunächst eine Verbindung mit Ihrer Instance her. Weitere Informationen finden Sie unter Connect to Your Instance im Amazon EC2 EC2-Benutzerhandbuch. Wenn Sie Probleme mit der Verbindung haben, finden Sie weitere Informationen unter Problembehandlung beim Herstellen einer Verbindung zu Ihrer Instance im Amazon EC2 EC2-Benutzerhandbuch.

So richten Sie den Agenten mithilfe der Amazon Linux AMI ein

Verwenden Sie den folgenden Befehl zum Herunterladen und Installieren des Agenten:

```
sudo yum install -y aws-kinesis-agent
```

So richten Sie den Agenten mit Red Hat Enterprise Linux ein

Verwenden Sie den folgenden Befehl zum Herunterladen und Installieren des Agenten:

```
sudo yum install -y https://s3.amazonaws.com/streaming-data-agent/aws-kinesis-agent-
latest.amzn2.noarch.rpm
```

Um den Agenten einzurichten mit GitHub

1. Laden Sie den Agenten von amazon-kinesis-agentawlabs/ herunter.
2. Installieren Sie den Agenten, indem Sie zum Download-Verzeichnis navigieren und den folgenden Befehl ausführen:

   ```
   sudo ./setup --install
   ```

So richten Sie den Agenten in einem Docker-Container ein

Kinesis Agent kann auch in einem Container über die amazonlinux-Containerbasis ausgeführt werden. Verwenden Sie die folgende Docker-Datei und führen Sie dann `docker build` aus.

```
FROM amazonlinux

RUN yum install -y aws-kinesis-agent which findutils
COPY agent.json /etc/aws-kinesis/agent.json

CMD ["start-aws-kinesis-agent"]
```

Konfigurieren und starten Sie den Agenten

So konfigurieren und starten Sie den Agenten

1. Öffnen und bearbeiten Sie die Konfigurationsdatei (als Superuser, wenn Sie standardmäßige Dateizugriffsberechtigungen nutzen): `/etc/aws-kinesis/agent.json`

Geben Sie in der Konfigurationsdatei die Dateien (`"filePattern"`) an, aus denen der Agent Daten sammelt, sowie den Namen des Streams (`"kinesisStream"`, an den der Agent die Daten sendet. Beachten Sie, dass der Dateiname ein Muster ist und der Agent Dateirotationen erkennt. Sie können nur einmal pro Sekunde Dateien rotieren oder neue Dateien erstellen. Der Agent verwendet den Zeitstempel der Dateierstellung, um die Dateien zu ermitteln, die gesammelt und zum Stream hinzugefügt werden. Wenn häufiger als einmal pro Sekunde Dateien rotiert oder neue Dateien erstellt werden, kann der Agent nicht mehr ordnungsgemäß zwischen den Dateien unterscheiden.

```
{
   "flows": [
        {
            "filePattern": "/tmp/app.log*",
            "kinesisStream": "yourkinesisstream"
        }
   ]
}
```

2. Starten Sie den Agenten manuell:

```
sudo service aws-kinesis-agent start
```

3. (Optional) Konfigurieren Sie den Agenten so, dass er beim Startup des Systems gestartet wird:

```
sudo chkconfig aws-kinesis-agent on
```

Der Agent wird jetzt als Systemdienst im Hintergrund ausgeführt. Er überwacht kontinuierlich die angegebenen Dateien und sendet Daten an den angegebenen Stream. Die Agentenaktivität wird in `/var/log/aws-kinesis-agent/aws-kinesis-agent.log` protokolliert.

Geben Sie die Einstellungen für die Agentenkonfiguration an

Der Agent unterstützt die beiden obligatorischen Konfigurationseinstellungen `filePattern` und `kinesisStream` sowie optionale Konfigurationseinstellungen für zusätzliche Funktionen. Sie können sowohl die obligatorische als auch die optionale Konfiguration in `/etc/aws-kinesis/agent.json` angeben.

Wenn Sie die Konfigurationsdatei ändern, müssen Sie den Agenten mit den folgenden Befehlen anhalten und starten:

```
sudo service aws-kinesis-agent stop
sudo service aws-kinesis-agent start
```

Alternativ können Sie auch den folgenden Befehl nutzen:

```
sudo service aws-kinesis-agent restart
```

Im Folgenden finden Sie die allgemeinen Konfigurationseinstellungen.

Konfigurationseinstellung	Description
`assumeRoleARN`	Der ARN der Rolle, die vom Benutzer übernommen werden muss. Weitere Informationen finden Sie unter AWS Kontenübergreifendes Delegieren des Zugriffs mithilfe von IAM-Rollen im IAM-Benutzerhandbuch.
`assumeRoleExternalId`	Eine optionale Kennung, die festlegt, wer die Rolle übernehmen kann. Weitere Informationen finden Sie unter Verwendung einer externen ID im IAM-Benutzerhandbuch.
`awsAccessKeyId`	AWS Zugriffsschlüssel-ID, die die Standardanmeldedaten überschreibt. Diese Einstellung hat Vorrang vor allen anderen Anbietern von Anmeldeinformationen.
`awsSecretAccessKey`	AWS geheimer Schlüssel, der die Standardanmeldedaten überschreibt. Diese Einstellung hat Vorrang vor allen anderen Anbietern von Anmeldeinformationen.
`cloudwatch.emitMetrics`	Ermöglicht dem Agenten, Metriken auszusenden, CloudWatch sofern diese Einstellung gesetzt ist (true). Standard: true
`cloudwatch.endpoint`	Der regionale Endpunkt für CloudWatch.

Konfigurationseinstellung	Description
	Standard: `monitoring.us-east-1.amazonaws.com`
`kinesis.endpoint`	Der regionale Endpunkt für Kinesis Data Streams. Standard: `kinesis.us-east-1.amazonaws.com`

Im Folgenden finden Sie die Konfigurationseinstellungen für den Ablauf.

Konfigurationseinstellung	Description
`dataProcessingOptions`	Die Liste der Verarbeitungsoptionen, die auf jeden analysierten Datensatz angewendet werden, ehe ein Datensatz an den Stream gesendet wird. Die Verarbeitungsoptionen werden in der angegebenen Reihenfolge ausgeführt. Weitere Informationen finden Sie unter Verwenden Sie den Agenten zur Vorverarbeitung von Daten.
`kinesisStream`	[Erforderlich] Der Name des Streams.
`filePattern`	[Erforderlich] Das Verzeichnis- und Dateimuster, das abgeglichen werden muss, damit es vom Agenten abgerufen werden kann. Für alle Dateien, die mit diesem Muster übereinstimmen, müssen Leseberechtigungen für `aws-kinesis-agent-user` vergeben werden. Für das die Dateien enthaltende Verzeichnis müssen Lese- und Ausführberechtigungen für `aws-kinesis-agent-user` vergeben werden.
`initialPosition`	Die Position, an der mit der Analyse der Datei begonnen wurde. Gültige Werte sind `START_OF_FILE` und `END_OF_FILE` . Standard: `END_OF_FILE`
`maxBufferAgeMillis`	Die maximale Zeit (in Millisekunden), während der durch den Agenten Daten gepuffert werden, bevor sie an den Stream gesendet werden. Wertebereich: 1.000 bis 900.000 (1 Sekunde bis 15 Minuten)

Konfigurationseinstellung	Description
	Standard: 60.000 (1 Minute)
`maxBufferSizeBytes`	Die maximale Größe (in Bytes), bis zu der durch den Agenten Daten gepuffert werden, bevor sie an den Stream gesendet werden. Wertebereich: 1 bis 4.194.304 (4 MB) Standard: 4.194.304 (4 MB)
`maxBufferSizeRecords`	Die maximale Anzahl der Datensätze, für die der Agent Daten puffert, ehe diese an den Stream gesendet werden. Wertebereich: 1 bis 500 Standard: 500
`minTimeBetweenFilePollsMillis`	Das Zeitintervall (in Millisekunden), in dem der Agent die überwachten Dateien auf neue Daten abfragt und analysiert. Wertbereich: 1 oder höher Standard: 100
`multiLineStartPattern`	Das Muster für die Identifizierung des Datensatzbeginns. Ein Datensatz besteht aus einer Zeile, die mit dem angegebenen Muster übereinstimmt, und allen folgenden Zeilen, die nicht dem Muster entsprechen. Gültige Werte sind reguläre Ausdrücke. Standardmäßig wird jede neue Zeile in den Protokolldateien als einziger Datensatz analysiert.
`partitionKeyOption`	Die Methode zum Erstellen des Partitionsschlüssels. Gültige Werte sind `RANDOM` (zufällig generierte Ganzzahl) und `DETERMINISTIC` (ein aus den Daten berechneter Hash-Wert). Standard: `RANDOM`

Konfigurationseinstellung	Description
`skipHeaderLines`	Die Anzahl der Zeilen, die der Agent überspringt, ehe mit der Analyse der überwachten Dateien begonnen wird. Wertbereich: 0 oder höher Standard: 0 (null)
`truncatedRecordTerminator`	Die Zeichenfolge, mit der der Agent einen analysierten Datensatz kürzt, wenn die Datensatzgröße das zulässige Datensatz-Limit von Kinesis Data Streams überschreitet. (1,000 KB) Standard: `'\n'` (Zeilenumbruch)

Überwachen Sie mehrere Dateiverzeichnisse und schreiben Sie in mehrere Streams

Wenn Sie mehrere Ablaufkonfigurationseinstellungen angeben, können Sie den Agenten so konfigurieren, dass er mehrere Dateiverzeichnisse überwacht und Daten an verschiedene Streams sendet. Im folgenden Konfigurationsbeispiel überwacht der Agent zwei Dateiverzeichnisse und sendet Daten an einen Kinesis-Stream bzw. einen Firehose-Lieferstream. Beachten Sie, dass Sie unterschiedliche Endpunkte für Kinesis Data Streams und Firehose angeben können, sodass sich Ihr Kinesis-Stream und Ihr Firehose-Lieferstream nicht in derselben Region befinden müssen.

```
{
    "cloudwatch.emitMetrics": true,
    "kinesis.endpoint": "https://your/kinesis/endpoint",
    "firehose.endpoint": "https://your/firehose/endpoint",
    "flows": [
        {
            "filePattern": "/tmp/app1.log*",
            "kinesisStream": "yourkinesisstream"
        },
        {
            "filePattern": "/tmp/app2.log*",
            "deliveryStream": "yourfirehosedeliverystream"
        }
```

```
    ]
}
```

Ausführlichere Informationen zur Verwendung des Agenten mit Firehose finden Sie unter Writing to Amazon Data Firehose with Kinesis Agent.

Verwenden Sie den Agenten zur Vorverarbeitung von Daten

Der Agent kann die Datensätze vorverarbeiten, die aus den überwachten Dateien analysiert wurden, ehe diese an Ihren Stream gesendet werden. Sie können dieses Feature aktivieren, indem Sie Ihrem Dateifluss die Konfigurationseinstellung `dataProcessingOptions` hinzufügen. Sie können eine oder mehrere Verarbeitungsoptionen hinzufügen. Diese werden in der angegebenen Reihenfolge ausgeführt.

Der Agent unterstützt die folgenden aufgelisteten Verarbeitungsoptionen. Der Agent ist ein Open-Source-Tool, sodass Sie dessen Verarbeitungsoptionen optimieren und erweitern können. Sie können den Agenten von Kinesis Agent herunterladen.

Verarbeitungsoptionen

`SINGLELINE`

Konvertiert einen mehrzeiligen Datensatz in einen einzeiligen Datensatz, indem Zeilenumbruchzeichen sowie vorangestellte und folgende Leerzeichen entfernt werden.

```
{
    "optionName": "SINGLELINE"
}
```

`CSVTOJSON`

Konvertiert einen Datensatz aus dem durch Trennzeichen getrennten Format in einen Datensatz im JSON-Format.

```
{
    "optionName": "CSVTOJSON",
    "customFieldNames": [ "field1", "field2", ... ],
    "delimiter": "yourdelimiter"
}
```

`customFieldNames`

[Erforderlich] Die Feldnamen, die als Schlüssel in den einzelnen JSON-Schlüssel-Wert-Paaren verwendet werden. Wenn Sie beispielsweise `["f1", "f2"]` angeben, wird der Datensatz „v1, v2“ in `{"f1":"v1","f2":"v2"}` konvertiert.

`delimiter`

Die Zeichenfolge, die als Trennzeichen im Datensatz verwendet wird. Standardmäßig wird ein Komma (,) verwendet.

`LOGTOJSON`

Konvertiert einen Datensatz aus einem Protokollformat in einen Datensatz im JSON-Format. Folgende Protokollformate werden unterstützt: Apache Common Log, Apache Combined Log, Apache Error Log und RFC3164 Syslog.

```
{
    "optionName": "LOGTOJSON",
    "logFormat": "logformat",
    "matchPattern": "yourregexpattern",
    "customFieldNames": [ "field1", "field2", … ]
}
```

`logFormat`

[Erforderlich] Das Format des Protokolleintrags. Folgende Werte sind möglich:

- `COMMONAPACHELOG` – Das Apache-Common-Log-Format. Jeder Protokolleintrag weist standardmäßig das folgende Muster auf: „`%{host} %{ident} %{authuser} [%{datetime}] \"%{request}\" %{response} %{bytes}`“.
- `COMBINEDAPACHELOG` – Das Apache-Combined-Log-Format. Jeder Protokolleintrag weist standardmäßig das folgende Muster auf: „`%{host} %{ident} %{authuser} [%{datetime}] \"%{request}\" %{response} %{bytes} %{referrer} %{agent}`“.
- `APACHEERRORLOG` – Das Apache-Error-Log-Format. Jeder Protokolleintrag weist standardmäßig das folgende Muster auf: „`[%{timestamp}] [%{module}: %{severity}] [pid %{processid}:tid %{threadid}] [client: %{client}] %{message}`“.

- SYSLOG— Das RFC3164 Syslog-Format. Jeder Protokolleintrag weist standardmäßig das folgende Muster auf: „`%{timestamp} %{hostname} %{program}[%{processid}]: %{message}`“.

`matchPattern`

Das reguläre Ausdrucksmuster, mit dem Werte aus den Protokolleinträgen extrahiert werden. Diese Einstellung wird verwendet, wenn Ihr Protokolleintrag nicht eines der vordefinierten Protokollformate aufweist. Bei dieser Einstellung müssen Sie auch `customFieldNames` angeben.

`customFieldNames`

Die benutzerdefinierten Feldnamen, die als Schlüssel in den einzelnen JSON-Schlüssel-Wert-Paaren verwendet werden. Mit dieser Einstellung können Sie Feldnamen für Werte definieren, die aus `matchPattern` extrahiert wurden, oder die Standardfeldnamen von vordefinierten Protokollformaten überschreiben.

Example: LOGTOJSON-Konfiguration

Nachfolgend ein Beispiel einer `LOGTOJSON`-Konfiguration für einen Apache Common Log-Eintrag, der in ein JSON-Format konvertiert wurde:

```
{
    "optionName": "LOGTOJSON",
    "logFormat": "COMMONAPACHELOG"
}
```

Vor der Konvertierung:

```
64.242.88.10 - - [07/Mar/2004:16:10:02 -0800] "GET /mailman/listinfo/hsdivision
 HTTP/1.1" 200 6291
```

Nach der Konvertierung:

```
{"host":"64.242.88.10","ident":null,"authuser":null,"datetime":"07/
Mar/2004:16:10:02 -0800","request":"GET /mailman/listinfo/hsdivision
 HTTP/1.1","response":"200","bytes":"6291"}
```

Example: LOGTOJSON-Konfiguration mit benutzerdefinierten Feldern

Im Folgenden ein weiteres Beispiel einer LOGTOJSON-Konfiguration:

```
{
    "optionName": "LOGTOJSON",
    "logFormat": "COMMONAPACHELOG",
    "customFieldNames": ["f1", "f2", "f3", "f4", "f5", "f6", "f7"]
}
```

Durch diese Konfigurationseinstellung wird der Apache Common Log-Eintrag aus dem vorherigen Beispiel wie folgt in ein JSON-Format konvertiert:

```
{"f1":"64.242.88.10","f2":null,"f3":null,"f4":"07/Mar/2004:16:10:02 -0800","f5":"GET /
mailman/listinfo/hsdivision HTTP/1.1","f6":"200","f7":"6291"}
```

Example: Konvertieren eines Apache Common Log-Eintrags

Bei der folgenden Ablaufkonfiguration wird ein Apache Common Log-Eintrag in einen einzeiligen Datensatz im JSON-Format umgewandelt:

```
{
    "flows": [
        {
            "filePattern": "/tmp/app.log*",
            "kinesisStream": "my-stream",
            "dataProcessingOptions": [
                {
                    "optionName": "LOGTOJSON",
                    "logFormat": "COMMONAPACHELOG"
                }
            ]
        }
    ]
}
```

Example: Konvertieren mehrzeiliger Datensätze

Bei der folgenden Ablaufkonfiguration werden mehrzeilige Datensätze analysiert, deren erste Zeile mit „[SEQUENCE=“ beginnt. Jeder Datensatz wird in einen einzeiligen Datensatz konvertiert. Anschließend werden Werte aus dem Datensatz basierend auf einem Tabulatortrennzeichen

extrahiert. Die extrahierten Werte werden zu angegebenen `customFieldNames`-Werten zugeordnet und ergeben so einen einzeiligen Datensatz im JSON-Format.

```
{
    "flows": [
        {
            "filePattern": "/tmp/app.log*",
            "kinesisStream": "my-stream",
            "multiLineStartPattern": "\\[SEQUENCE=",
            "dataProcessingOptions": [
                {
                    "optionName": "SINGLELINE"
                },
                {
                    "optionName": "CSVTOJSON",
                    "customFieldNames": [ "field1", "field2", "field3" ],
                    "delimiter": "\\t"
                }
            ]
        }
    ]
}
```

Example: LOGTOJSON-Konfiguration mit Übereinstimmungsmuster

Nachfolgend ein Beispiel einer `LOGTOJSON`-Konfiguration für einen Apache Common Log-Eintrag, der in das JSON-Format konvertiert wurde. Das letzte Feld (Bytes) wurde ausgelassen:

```
{
    "optionName": "LOGTOJSON",
    "logFormat": "COMMONAPACHELOG",
    "matchPattern": "^([\\d.]+) (\\S+) (\\S+) \\[([\\w:/]+\\s[+\\-]\\d{4})\\] \"(.
+?)\" (\\d{3})",
    "customFieldNames": ["host", "ident", "authuser", "datetime", "request",
 "response"]
}
```

Vor der Konvertierung:

```
123.45.67.89 - - [27/Oct/2000:09:27:09 -0400] "GET /java/javaResources.html HTTP/1.0"
 200
```

Nach der Konvertierung:

```
{"host":"123.45.67.89","ident":null,"authuser":null,"datetime":"27/Oct/2000:09:27:09
 -0400","request":"GET /java/javaResources.html HTTP/1.0","response":"200"}
```

Verwenden Sie CLI-Befehle für Agenten

Automatisches Startup des Agenten beim Systemstart:

```
sudo chkconfig aws-kinesis-agent on
```

Prüfen des Status des Agenten:

```
sudo service aws-kinesis-agent status
```

Beenden des Agenten:

```
sudo service aws-kinesis-agent stop
```

Auslesen der Protokolldatei des Agenten von diesem Speicherort:

```
/var/log/aws-kinesis-agent/aws-kinesis-agent.log
```

Deinstallieren des Agenten:

```
sudo yum remove aws-kinesis-agent
```

Häufig gestellte Fragen

Gibt es einen Kinesis Agent für Windows?

Kinesis Agent für Windows ist eine andere Software als Kinesis Agent für Linux-Plattformen.

Warum wird Kinesis Agent langsamer and/or **`RecordSendErrors`** und stärker?

Dies ist normalerweise auf die Drosselung durch Kinesis zurückzuführen. Überprüfen Sie die `WriteProvisionedThroughputExceeded` Metrik für Kinesis Data Streams oder die `ThrottledRecords` Metrik für Firehose Delivery Streams. Jede Erhöhung dieser Metriken von 0 zeigt an, dass die Stream-Grenzwerte erhöht werden müssen. Weitere Informationen finden Sie unter Kinesis-Datenstrom-Grenzwerte und Amazon Firehose Delivery Streams.

Sobald Sie die Drosselung ausgeschlossen haben, überprüfen Sie, ob der Kinesis Agent so konfiguriert ist, dass er eine große Menge kleiner Dateien durchsucht. Es gibt eine Verzögerung, wenn der Kinesis Agent eine neue Datei überwacht, daher sollte der Kinesis-Agent eine kleine Menge größerer Dateien überwachen. Versuchen Sie, Ihre Protokolldateien in größeren Dateien zusammenzufassen.

Warum erhalte ich **`java.lang.OutOfMemoryError`**-Ausnahmen?

Kinesis Agent verfügt nicht über genügend Arbeitsspeicher, um seinen aktuellen Workload zu bewältigen. Versuchen Sie, `JAVA_START_HEAP` und `JAVA_MAX_HEAP` in `/usr/bin/start-aws-kinesis-agent` zu erhöhen und den Agenten neu zu starten.

Warum erhalte ich **`IllegalStateException : connection pool shut down`**-Ausnahmen?

Kinesis Agent verfügt nicht über genügend Verbindungen, um seinen aktuellen Workload zu bewältigen. Versuchen Sie, `maxConnections` und `maxSendingThreads` in den allgemeinen Konfigurationseinstellungen des Agenten unter `/etc/aws-kinesis/agent.json` zu erhöhen. Der Standardwert für diese Felder ist das 12-fache der verfügbaren Laufzeitprozessoren. Weitere Informationen zu den Einstellungen für erweiterte Agentenkonfigurationen finden Sie unter AgentConfiguration.java.

Wie kann ich ein anderes Problem mit Kinesis Agent beheben?

`DEBUG`-Level-Protokolle können in `/etc/aws-kinesis/log4j.xml` aktiviert werden.

Wie sollte ich Kinesis Agent konfigurieren?

Je kleiner das `maxBufferSizeBytes`, desto häufiger sendet der Kinesis Agent Daten. Dies kann nützlich sein, da es die Lieferzeit von Datensätzen verkürzt, aber es erhöht auch die Anfragen pro Sekunde an Kinesis.

Warum sendet Kinesis Agent doppelte Datensätze?

Dies ist auf eine Fehlkonfiguration bei der Dateiüberwachung zurückzuführen. Stellen Sie sicher, dass jedes `fileFlow's filePattern` nur einer Datei entspricht. Dies kann auch auftreten, wenn der verwendete `logrotate`-Modus im `copytruncate`-Modus ist. Versuchen Sie, den Modus auf den Standard- oder Erstellungsmodus zu ändern, um Duplikate zu vermeiden. Weitere Informationen zum Umgang mit doppelten Datensätzen finden Sie unter Umgang mit doppelten Datensätzen.

Schreiben Sie mithilfe anderer AWS Dienste in Kinesis Data Streams

Die folgenden AWS Services können direkt in Amazon Kinesis Data Streams integriert werden, um Daten in Kinesis-Datenstreams zu schreiben. Überprüfen Sie die Informationen für jeden Service, an dem Sie interessiert sind, und verweisen Sie auf die bereitgestellten Referenzen.

Themen

- Schreiben Sie in Kinesis Data Streams mit AWS Amplify
- Schreiben Sie mit Amazon Aurora in Kinesis Data Streams
- Schreiben Sie mit Amazon in Kinesis Data Streams CloudFront
- Schreiben Sie mithilfe von Amazon CloudWatch Logs in Kinesis Data Streams
- Schreiben Sie mit Amazon Connect in Kinesis Data Streams
- Schreiben Sie in Kinesis Data Streams mit AWS Database Migration Service
- Schreiben Sie mit Amazon DynamoDB in Kinesis Data Streams
- Schreiben Sie mit Amazon in Kinesis Data Streams EventBridge
- Schreiben Sie in Kinesis Data Streams mit AWS IoT Core
- Schreiben Sie mit Amazon Relational Database Service in Kinesis Data Streams
- Schreiben Sie mit Amazon Pinpoint in Kinesis Data Streams
- Schreiben Sie mithilfe der Amazon Quantum Ledger Database (Amazon QLDB) in Kinesis Data Streams

Schreiben Sie in Kinesis Data Streams mit AWS Amplify

Sie können Amazon Kinesis Data Streams verwenden, um Daten aus Ihren mit AWS Amplify erstellten mobilen Anwendungen für die Echtzeitverarbeitung zu streamen. Sie können dann Echtzeit-Dashboards erstellen, Ausnahmen erfassen, Warnungen generieren, Empfehlungen erzeugen und in Echtzeit andere geschäftliche oder betriebliche Entscheidungen treffen. Sie können Daten auch an andere Dienste wie Amazon Simple Storage Service, Amazon DynamoDB und Amazon Redshift senden.

Weitere Informationen finden Sie unter Verwenden von Amazon Kinesis im AWS Amplify Developer Center.

Schreiben Sie mit Amazon Aurora in Kinesis Data Streams

Sie können Amazon Kinesis Data Streams verwenden, um Aktivitäten auf Ihren Amazon Aurora-DB-Clustern zu überwachen. Mithilfe von Database Activity Streams überträgt Ihr Aurora-DB-Cluster Aktivitäten in Echtzeit an einen Amazon Kinesis Data Streams. Anschließend können Sie Anwendungen für das Compliance-Management entwickeln, die diese Aktivitäten nutzen, prüfen und Warnmeldungen generieren. Sie können auch Amazon Firehose verwenden, um die Daten zu speichern.

Weitere Informationen finden Sie unter Datenbankaktivitätsstreams im Entwicklerhandbuch zu Amazon Aurora.

Schreiben Sie mit Amazon in Kinesis Data Streams CloudFront

Sie können Amazon Kinesis Data Streams mit CloudFront Echtzeitprotokollen verwenden und Informationen über Anfragen an eine Distribution in Echtzeit abrufen. Sie können dann Ihren eigenen Kinesis Data Stream Consumer erstellen oder Amazon Data Firehose verwenden, um die Protokolldaten an Amazon S3, Amazon Redshift, Amazon OpenSearch Service oder einen Protokollverarbeitungsservice eines Drittanbieters zu senden.

Weitere Informationen finden Sie unter Echtzeitprotokolle im Amazon CloudFront Developer Guide.

Schreiben Sie mithilfe von Amazon CloudWatch Logs in Kinesis Data Streams

Sie können CloudWatch Abonnements verwenden, um Zugriff auf einen Echtzeit-Feed mit Protokollereignissen von Amazon CloudWatch Logs zu erhalten und diesen zur Verarbeitung, Analyse und zum Laden in andere Systeme an einen Kinesis-Datenstream senden zu lassen.

Weitere Informationen finden Sie unter Echtzeitverarbeitung von Protokolldaten mit Abonnements im Amazon CloudWatch Logs-Benutzerhandbuch.

Schreiben Sie mit Amazon Connect in Kinesis Data Streams

Sie können Kinesis Data Streams verwenden, um Kontaktdatensätze und Agentenereignisse in Echtzeit aus Ihrer Amazon-Connect-Instance zu exportieren. Sie können auch Datenstreaming von Amazon Connect Connect-Kundenprofilen aktivieren, um automatisch Updates für einen Kinesis-Datenstream über die Erstellung neuer Profile oder Änderungen an bestehenden Profilen zu erhalten.

Anschließend können Sie Verbraucheranwendungen erstellen, um die Daten in Echtzeit zu verarbeiten und zu analysieren. Mithilfe von Kontaktdatensätzen und Kundenprofildaten können Sie beispielsweise Ihre Quellsystemdaten, wie CRMs z. B. Marketingautomatisierungstools, auf up-to-date dem neuesten Stand halten. Mithilfe der Ereignisdaten der Agenten können Sie Dashboards erstellen, die Agenteninformationen und Ereignisse anzeigen und benutzerdefinierte Benachrichtigungen über bestimmte Agentenaktivitäten auslösen.

Weitere Informationen finden Sie unter Datenstreaming für Ihre Instance, Echtzeit-Export einrichten und Agenten-Event-Streams im Amazon-Connect-Administratorhandbuch.

Schreiben Sie in Kinesis Data Streams mit AWS Database Migration Service

Sie können AWS Database Migration Service es verwenden, um Daten in einen Kinesis-Datenstream zu migrieren. Anschließend können Sie Verbraucheranwendungen erstellen, die die Datensätze in Echtzeit verarbeiten. Sie können Daten auch problemlos nachgelagert an andere Services wie Amazon Simple Storage Service, Amazon DynamoDB und Amazon Redshift senden

Weitere Informationen finden Sie unter Nutzung von Kinesis Data Streams im Benutzerhandbuch für AWS Database Migration Service .

Schreiben Sie mit Amazon DynamoDB in Kinesis Data Streams

Sie können Amazon Kinesis Data Streams verwenden, um Änderungen an Amazon DynamoDB zu erfassen. Kinesis Data Streams erfasst Änderungen auf Elementebene in jeder DynamoDB-Tabelle und repliziert sie in einen Kinesis Data Stream. Ihre Verbraucheranwendungen können auf diesen Stream zugreifen, um Änderungen auf Artikelebene in Echtzeit anzuzeigen und diese Änderungen anschließend weiterzuleiten oder auf der Grundlage des Inhalts Maßnahmen zu ergreifen.

Weitere Informationen finden Sie unter wie Kinesis Data Streams mit DynamoDB funktionieren im Amazon-DynamoDB-Entwicklerhandbuch.

Schreiben Sie mit Amazon in Kinesis Data Streams EventBridge

Mit Kinesis Data Streams können Sie AWS API-Aufrufereignisse EventBridge an einen Stream senden, Verbraucheranwendungen erstellen und große Datenmengen verarbeiten. Sie können Kinesis Data Streams auch als Ziel in EventBridge Pipes verwenden und Datensätze nach optionaler Filterung und Anreicherung als Stream aus einer der verfügbaren Quellen bereitstellen.

Weitere Informationen finden Sie unter Senden von Ereignissen an einen Amazon Kinesis Kinesis-Stream und EventBridge Pipes im EventBridge Amazon-Benutzerhandbuch.

Schreiben Sie in Kinesis Data Streams mit AWS IoT Core

Mithilfe von IoT-Regelaktionen können Sie Daten in Echtzeit aus MQTT-Nachrichten in AWS AWS IoT Core schreiben. Anschließend können Sie Anwendungen erstellen, die die Daten verarbeiten, ihren Inhalt analysieren und Warnmeldungen generieren und sie an Analyseanwendungen oder andere AWS -Services weiterleiten.

Weitere Informationen finden Sie unter Kinesis Data Streams im AWS IoT Core-Entwicklerhandbuch.

Schreiben Sie mit Amazon Relational Database Service in Kinesis Data Streams

Sie können Amazon Kinesis Data Streams verwenden, um Aktivitäten auf Ihren Amazon-RDS-Instances zu überwachen. Mithilfe von Database Activity Streams überträgt Amazon RDS Aktivitäten in Echtzeit in einen Kinesis-Datenstream. Anschließend können Sie Anwendungen für das Compliance-Management entwickeln, die diese Aktivitäten nutzen, prüfen und Warnmeldungen generieren. Sie können auch Amazon Data Firehose verwenden, um die Daten zu speichern.

Weitere Informationen finden Sie unter Datenbankaktivitätsstreams im Entwicklerhandbuch zu Amazon RDS.

Schreiben Sie mit Amazon Pinpoint in Kinesis Data Streams

Sie können Amazon Pinpoint so einrichten, dass Ereignisdaten an Amazon Kinesis Data Streams gesendet werden. Amazon Pinpoint kann Ereignisdaten für Kampagnen, Reisen und transaktionale E-Mail- und SMS-Nachrichten senden. Sie können die Daten dann in Analyseanwendungen aufnehmen oder Ihre eigenen Verbraucheranwendungen erstellen, die auf der Grundlage des Inhalts der Ereignisse Maßnahmen ergreifen.

Weitere Informationen finden Sie unter Streaming-Events im Entwicklerhandbuch zu Amazon Pinpoint.

Schreiben Sie mithilfe der Amazon Quantum Ledger Database (Amazon QLDB) in Kinesis Data Streams

Sie können in Amazon QLDB einen Stream erstellen, der jede Dokumentrevision erfasst, die in Ihr Journal übernommen wurde, und diese Daten in Echtzeit an Amazon Kinesis Data Streams

übermittelt. Ein QLDB-Stream ist ein kontinuierlicher Datenfluss aus dem Ledger-Journal zu einer Kinesis-Datenstromressource. Anschließend können Sie die Kinesis-Streaming-Plattform oder die Kinesis-Client-Bibliothek verwenden, um Ihren Stream zu nutzen, die Datensätze zu verarbeiten und den Dateninhalt zu analysieren. Ein QLDB-Stream schreibt Ihre Daten in drei Datensatztypen in Kinesis Data Streams: `control`, `block summary` und `revision details`.

Weitere Informationen finden Sie unter Streams im Amazon-QLDB-Entwicklerhandbuch.

Schreiben Sie mithilfe von Integrationen von Drittanbietern in Kinesis Data Streams

Sie können Daten mit einer der folgenden Drittanbieteroptionen, die in Kinesis Data Streams integriert sind, in Kinesis Data Streams schreiben. Wählen Sie die Option aus, über die Sie mehr erfahren möchten, und finden Sie Ressourcen und Links zu relevanter Dokumentation.

Themen

- Apache Flink
- Fluentd
- Debezium
- Orakel GoldenGate
- Kafka Connect
- Adobe Experience
- Striim

Apache Flink

Apache Flink ist ein Framework und eine verteilte Verarbeitungs-Engine für statusbehaftete Berechnungen über unbegrenzte und begrenzte Datenströme. Weitere Informationen zum Schreiben von Apache Flink in Kinesis Data Streams finden Sie unter Amazon Kinesis Data Streams Connector.

Fluentd

Fluentd ist ein Open-Source-Datensammler für eine einheitliche Protokollierungsebene. Weitere Informationen zum Schreiben von Fluentd in Kinesis Data Streams. Weitere Informationen finden Sie unter Stream-Verarbeitung mit Kinesis.

Debezium

Debezium ist eine verteilte Open-Source-Plattform für die Erfassung von Änderungsdaten. Weitere Informationen zum Schreiben von Debezium in Kinesis Data Streams finden Sie unter Streaming von MySQL-Datenänderungen zu Amazon Kinesis.

Orakel GoldenGate

Oracle GoldenGate ist ein Softwareprodukt, mit dem Sie Daten von einer Datenbank in eine andere replizieren, filtern und transformieren können. Weitere Informationen zum Schreiben von Oracle GoldenGate in Kinesis Data Streams finden Sie unter Datenreplikation nach Kinesis Data Stream mit Oracle. GoldenGate

Kafka Connect

Kafka Connect ist ein Tool für skalierbares und zuverlässiges Streamen von Daten zwischen Apache Kafka und anderen Systemen. Weitere Informationen zum Schreiben von Apache Kafka in Kinesis Data Streams finden Sie unter der Kinesis-Kafka-Konnektor.

Adobe Experience

Die Adobe Experience Platform ermöglicht es Unternehmen, Kundendaten aus jedem System zu zentralisieren und zu standardisieren. Anschließend werden Datenwissenschaft und Machine Learning angewendet, um das Design und die Bereitstellung umfassender, personalisierter Erlebnisse erheblich zu verbessern. Weitere Informationen zum Schreiben von Daten von der Adobe Experience Platform in Kinesis Data Streams finden Sie unter So erstellen Sie eine Amazon-Kinesis-Verbindung.

Striim

Striim ist eine vollständige end-to-end In-Memory-Plattform zum Sammeln, Filtern, Transformieren, Anreichern, Aggregieren, Analysieren und Bereitstellen von Daten in Echtzeit. Weitere Informationen zum Schreiben von Daten von Striim in Kinesis Data Streams finden Sie im Kinesis Writer.

Problembehandlung Amazon Kinesis Data Streams Streams-Produzenten

Die folgenden Themen bieten Lösungen für häufig auftretende Probleme mit Amazon Kinesis Data Streams Streams-Herstellern:

- Meine Producer-Anwendung schreibt langsamer als erwartet
- Ich erhalte einen unautorisierten KMS-Masterkey-Berechtigungsfehler
- Beheben Sie andere häufig auftretende Probleme von Herstellern

Meine Producer-Anwendung schreibt langsamer als erwartet

Die häufigsten Gründe dafür, dass der Schreibdurchsatz langsamer als erwartet ist, sind:

- Die Servicelimits wurden überschritten
- Ich möchte meinen Producer optimieren
- Missbrauch von flushSync() Vorgängen

Die Servicelimits wurden überschritten

Wenn Sie wissen möchten, ob die Service Limits überschritten werden, prüfen Sie, ob Ihr Datenproduzent Ausnahmen im Service auslöst, und validieren Sie, welche API-Operationen gedrosselt werden. Beachten Sie, dass es je nach Aufruf verschiedene Limits gibt, siehe Kontingente und -Einschränkungen. Beispielsweise gelten zusätzlich zu den bekannten Limits für Schreib- und Lesevorgänge auf Shard-Ebene folgenden Limits auf Stream-Ebene:

- CreateStream
- DeleteStream
- ListStreams
- GetShardIterator
- MergeShards
- DescribeStream
- DescribeStreamSummary

Die Operationen `CreateStreamDeleteStream`, `ListStreams`, `GetShardIterator` und `MergeShards` dürfen maximal 5 mal pro Sekunde aufgerufen werden. Die `DescribeStream`-Operation ist auf 10 Aufrufe pro Sekunde begrenzt. Die `DescribeStreamSummary`-Operation ist auf 20 Aufrufe pro Sekunde begrenzt.

Wenn diese Aufrufe nicht das Problem sind, stellen Sie sicher, dass Sie einen Partitionsschlüssel ausgewählt haben, mit dem Sie die PUT-Operationen gleichmäßig auf alle Shards verteilen können,

und dass Sie keinen Partitionsschlüssel haben, der die Service Limits ausschöpft, wenn der Rest dies nicht tut. Dazu müssen Sie den Spitzendurchsatz messen und die Anzahl der Shards im Stream berücksichtigen. Weitere Informationen zum Verwalten von Streams finden Sie unter Kinesis-Datenstreams erstellen und verwalten.

> **Tip**
>
> Bitte beachten, dass Sie bei der Berechnung der Durchsatzdrosselung auf den nächsten Byte aufrunden müssen, wenn Sie die Einzeldatensatz-Operation PutRecord verwenden. Bei der Mehrfachdatensatz-Operation PutRecords wird die kumulative Anzahl der Datensätze in den einzelnen Aufrufen gerundet. Eine `PutRecords`-Anforderung mit 600 Datensätzen und einer Größe von 1,1 KB wird beispielsweise nicht gedrosselt.

Ich möchte meinen Producer optimieren

Bevor Sie mit der Optimierung Ihres Producers beginnen, müssen Sie die folgenden wichtigen Aufgaben erledigen. Bestimmen Sie zunächst den gewünschten Spitzenwert hinsichtlich Datensatzgröße und Datensätze pro Sekunde. Schließen Sie dann die Stream-Kapazität als Begrenzungsfaktor aus (Die Servicelimits wurden überschritten). Nutzen Sie die folgenden Tipps zur Fehlerbehebung und Optimierungsanleitungen für die beiden häufigsten Produzententypen, wenn Sie die Stream-Kapazität als Begrenzungsfaktor ausgeschlossen haben.

Großer Produzent

Ein großer Produzent wird in der Regel von einem lokalen Server oder einer Amazon-EC2-Instance ausgeführt. Für Kunden, die einen höheren Durchsatz eines großen Produzenten benötigen, ist in der Regel die Latenz pro Datensatz entscheidend. Zu den Strategien für den Umgang mit Latenz gehören die folgenden: Wenn der Kunde Datensätze per Mikro-Batch/Puffer speichern kann, verwendet er die Amazon Kinesis Producer Library (die über eine erweiterte Aggregationslogik verfügt), die Operation mit mehreren Datensätzen oder aggregieren Sie Datensätze zu einer größeren Datei PutRecords, bevor Sie den Einzeldatensatzvorgang verwenden. PutRecord Wenn Sie keine Stapel bilden und keine Pufferung verwenden können, nutzen Sie mehrere Threads, um gleichzeitig in Service Kinesis Data Streams schreiben zu können. Zu diesen AWS SDK für Java und anderen SDKs gehören asynchrone Clients, die dies mit sehr wenig Code bewerkstelligen können.

Kleiner Produzent

Eine kleiner Produzent ist in der Regel eine mobile App, ein IoT-Gerät oder ein Webclient. Wenn es sich um eine mobile App handelt, empfehlen wir, den `PutRecords` Vorgang oder den Kinesis Recorder auf dem AWS Handy SDKs zu verwenden. Weitere Informationen finden Sie in den Handbüchern „ AWS Mobile SDK für Android Erste Schritte“ und „ AWS Mobile SDK for iOS Erste Schritte“. Mobile Anwendungen müssen Verbindungsunterbrechungen bewältigen und benötigen eine Art Stapeleingabe, beispielsweise `PutRecords`. Wenn Sie aus irgendeinem Grund keine Stapel bilden können, lesen Sie sich die oben stehenden Informationen zu großen Produzenten durch. Ist der Produzent ein Browser, ist die Menge der generierten Dateien in der Regel sehr klein. Allerdings platzieren Sie PUT-Operationen im kritischen Pfad der Anwendung. Dies wird von uns nicht empfohlen.

Missbrauch von **`flushSync()`** Vorgängen

Eine `flushSync()` falsche Verwendung kann die Schreibleistung erheblich beeinträchtigen. Der `flushSync()` Vorgang ist für Shutdown-Szenarien konzipiert, um sicherzustellen, dass alle gepufferten Datensätze gesendet werden, bevor die KPL-Anwendung beendet wird. Wenn Sie diesen Vorgang nach jedem Schreibvorgang implementieren, kann dies zu einer erheblichen zusätzlichen Latenz von etwa 500 ms pro Schreibvorgang führen. Stellen Sie sicher, dass Sie die Implementierung `flushSync()` nur für das Herunterfahren der Anwendung vorgenommen haben, um unnötige zusätzliche Verzögerungen bei der Schreibleistung zu vermeiden.

Ich erhalte einen unautorisierten KMS-Masterkey-Berechtigungsfehler

Dieser Fehler tritt auf, wenn eine Produzentenanwendung ohne Berechtigung für den KMS-Masterschlüssel Daten in einen verschlüsselten Stream schreibt. Informationen zum Zuweisen von Berechtigungen zu einer Anwendung für den Zugriff auf einen KMS-Schlüssel finden Sie unter Verwenden von Schlüsselrichtlinien in AWS KMS und Verwenden von IAM-Richtlinien mit AWS KMS.

Beheben Sie andere häufig auftretende Probleme von Herstellern

- Warum gibt mein Kinesis-Datenstrom einen 500 Internal Server Error zurück?
- Wie behebe ich Timeout-Fehler beim Schreiben von Flink in Kinesis Data Streams?
- Wie behebe ich Drosselungsfehler in Kinesis Data Streams?
- Warum drosselt mein Kinesis-Datenstrom?
- Wie kann ich Datensätze mithilfe der KPL in einen Kinesis-Datenstrom einfügen?

Optimieren Sie Kinesis Data Streams Streams-Produzenten

Sie können Ihre Amazon Kinesis Data Streams Streams-Producer je nach dem spezifischen Verhalten, das Sie beobachten, weiter optimieren. Sehen Sie sich die folgenden Themen an, um Lösungen zu finden.

Themen

- Passen Sie KPL-Wiederholungsversuche und das Verhalten bei der Ratenbegrenzung an
- Wenden Sie bewährte Methoden auf die KPL-Aggregation an

Passen Sie KPL-Wiederholungsversuche und das Verhalten bei der Ratenbegrenzung an

Wenn Sie Benutzerdatensätze der Amazon Kinesis Producer Library (KPL) mithilfe des `addUserRecord()` KPL-Vorgangs hinzufügen, erhält ein Datensatz einen Zeitstempel und wird einem Puffer hinzugefügt, dessen Frist durch den `RecordMaxBufferedTime` Konfigurationsparameter festgelegt wird. Diese stamp/deadline Zeitkombination legt die Pufferpriorität fest. Datensätze werden aufgrund der folgenden Kriterien aus dem Puffer entfernt:

- Puffer-Priorität
- Aggregierungskonfiguration
- Sammlungskonfiguration

Die Parameter der Aggregierungskonfiguration und der Sammlungskonfiguration, die das Pufferverhalten beeinflussen, sind:

- `AggregationMaxCount`
- `AggregationMaxSize`
- `CollectionMaxCount`
- `CollectionMaxSize`

Die gelöschten Datensätze werden dann mithilfe eines Aufrufs der API-Operation `PutRecords` von Kinesis Data Streams als Datensätze von Amazon Kinesis Data Streams an Ihren Kinesis-Datenstrom gesendet. Die Operation `PutRecords` sendet Anfragen an Ihre Stream, die gelegentlich vollständige oder teilweise Fehlschläge zeigen. Fehlgeschlagene Datensätze werden automatisch

wieder dem KPL-Puffer hinzugefügt. Die neue Frist wird auf der Grundlage des kleineren dieser beiden Werte festgelegt:

- Die Hälfte der aktuellen `RecordMaxBufferedTime`-Konfiguration
- Der time-to-live Wert des Datensatzes

Mit dieser Strategie können erneut versuchte KPL-Benutzerdatensätze in nachfolgende Kinesis Data Streams Streams-API-Aufrufe aufgenommen werden, um den Durchsatz zu verbessern und die Komplexität zu reduzieren und gleichzeitig den Wert des Kinesis Data Streams Streams-Datensatzes durchzusetzen. time-to-live Es gibt keinen Backoff-Algorithmus, was dies zu einer relativ aggressiven Wiederholungsstrategie macht. Spamming aufgrund übermäßiger Wiederholungen wird durch die Quotengrenze bewirkt, die im nächsten Abschnitt erläutert wird.

Ratenbegrenzung

Das KPL enthält eine Quotenbegrenzungsfunktion, die den pro Shard von einem einzelnen Producer gesendeten Durchsatz begrenzt. Die Quotenbegrenzung wird durch einen Token-Bucket-Algorithmus mit separaten Buckets für Datensätze von Kinesis Data Streams und Bytes implementiert. Jeder erfolgreiche Schreibvorgang in einen Kinesis-Datenstrom fügt jedem Bucket ein Token (oder mehrere Token) hinzu, bis ein bestimmter Grenzwert erreicht ist. Dieser Grenzwert ist konfigurierbar, ist aber standardmäßig um 50 % höher als das tatsächliche Shard-Limit eingestellt, um die Shard-Sättigung durch einen einzelnen Produzenten zu ermöglichen.

Sie können diesen Grenzwert herabsetzen, um Spamming durch übermäßig viele Wiederholungsversuche zu reduzieren. Die bewährte Methode besteht jedoch darin, dass jeder Produzent aggressiv einen maximalen Durchsatz durch Wiederholungen zu erzielen versucht und mit der daraus resultierenden, als übermäßig angesehenen Drosselung so umzugehen, dass die Kapazität des Streams erweitert und eine geeignete Partitionsschlüsselstrategie implementiert wird.

Wenden Sie bewährte Methoden auf die KPL-Aggregation an

Das Sequenznummernschema der resultierenden Amazon Kinesis Data Streams Streams-Datensätze bleibt zwar gleich, aber die Aggregation führt dazu, dass die Indizierung der Benutzerdatensätze der Amazon Kinesis Producer Library (KPL), die in einem aggregierten Kinesis Data Streams Streams-Datensatz enthalten sind, bei 0 (Null) beginnt. Solange Sie sich jedoch nicht auf Sequenznummern verlassen, um Ihre KPL-Benutzerdatensätze eindeutig zu identifizieren, kann Ihr Code dies ignorieren, da die Aggregation (Ihrer KPL-Benutzerdatensätze) zu einer Kinesis Data Streams (Datensatz) und anschließende Deaggregation (eines Kinesis Data Streams-Datensatzes

in Ihre KPL-Benutzerdatensätze) erledigt das automatisch für Sie. Dies gilt unabhängig davon, ob Ihr Kunde die KCL oder das AWS SDK verwendet. Um diese Aggregationsfunktion nutzen zu können, müssen Sie den Java-Teil der KPL in Ihren Build übernehmen, wenn Ihr Consumer mithilfe der im SDK bereitgestellten API geschrieben wurde. AWS

Wenn Sie beabsichtigen, Folgenummern als einzige Kennungen Ihrer KPL-Benutzerdatensätze zu verwenden, empfehlen wir die Verwendung der vertragsgemäßen Operationen `public int hashCode()` und `public boolean equals(Object obj)` in `Record` und `UserRecord` zur Aktivierung des Vergleichs Ihrer KPL-Benutzerdatensätze. Wenn Sie zudem die Teilsequenznummer des KPL;-Benutzerdatensatzes untersuchen möchten, können Sie eine Typumwandlung in eine `UserRecord`-Instance vornehmen und deren Teilsequenznummer abrufen.

Weitere Informationen finden Sie unter Implementieren Sie die Deaggregation für Verbraucher.

Daten aus Amazon Kinesis Data Streams lesen

Ein Konsument ist eine Anwendung, die alle Daten aus einem Kinesis-Datenstrom verarbeitet. Wenn ein Verbraucher Enhanced Fan-Out verwendet, erhält er seinen eigenen Lesedurchsatz MB/sec von 2 zugeteilt, sodass mehrere Verbraucher Daten aus demselben Stream parallel lesen können, ohne mit anderen Verbrauchern um den Lesedurchsatz zu kämpfen. Informationen zum Verwenden der Shard-Funktionen für erweiterte Rundsendungen erhalten Sie unter Entwickeln Sie verbesserte Fan-Out-Verbraucher mit dediziertem Durchsatz.

Sie können Consumer für Kinesis Data Streams mithilfe der Kinesis Client Library (KCL) oder erstellen. AWS SDK für Java Sie können Kunden auch mithilfe anderer AWS Dienste wie AWS Lambda Amazon Managed Service für Apache Flink und Amazon Data Firehose gewinnen. Kinesis Data Streams unterstützt Integrationen mit anderen AWS Diensten wie Amazon EMR, Amazon und Amazon Redshift. Es unterstützt auch Integrationen von Drittanbietern wie Apache Flink AWS Glue, Adobe Experience Platform, Apache Druid, Apache Spark, Databricks, Confluent Platform, Kinesumer und Talend. EventBridge

Themen

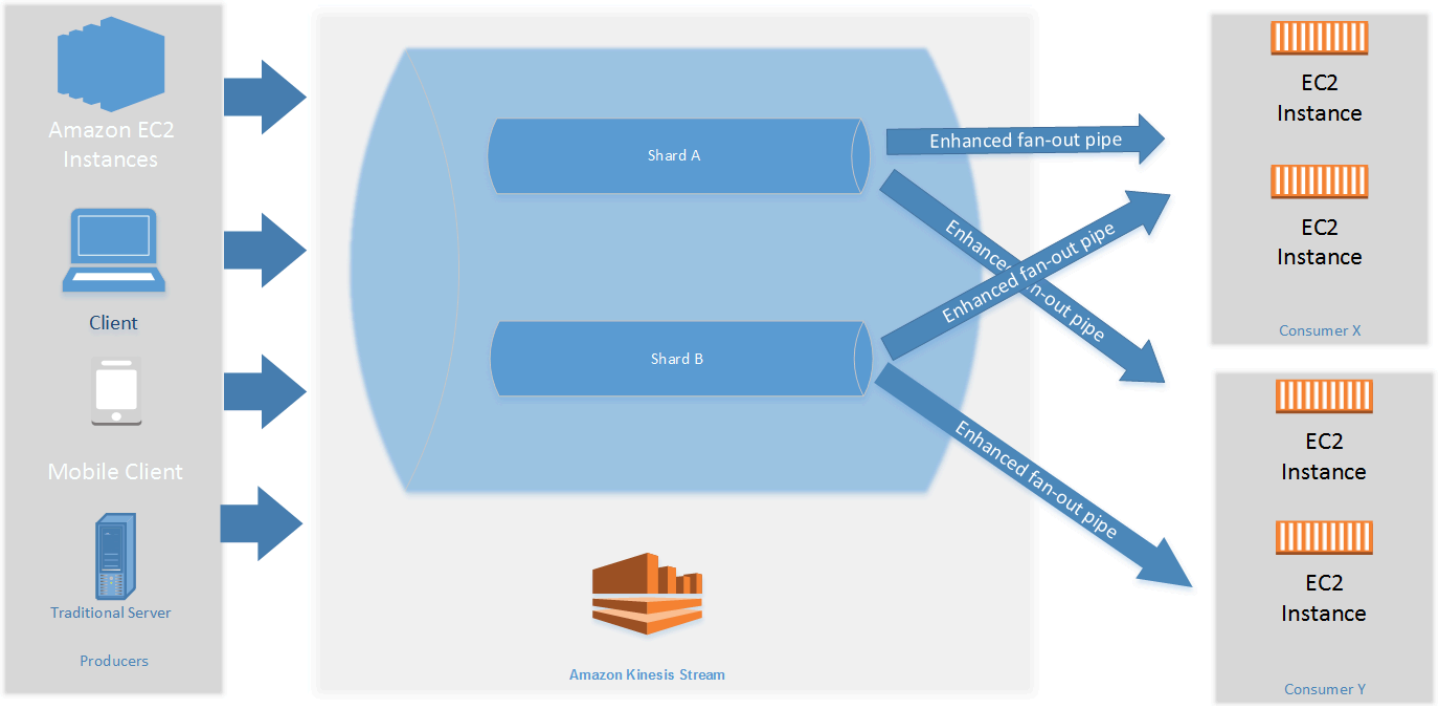
- Entwickeln Sie verbesserte Fan-Out-Verbraucher mit dediziertem Durchsatz
- Verwenden Sie den Data Viewer in der Kinesis-Konsole
- Fragen Sie Ihre Datenströme in der Kinesis-Konsole ab
- Verwenden Sie die Kinesis-Clientbibliothek
- Entwickeln Sie Verbraucher mit dem AWS SDK für Java
- Entwickeln Sie Verbraucher mit AWS Lambda
- Entwickeln Sie Kunden mithilfe von Amazon Managed Service für Apache Flink
- Entwickeln Sie Verbraucher mithilfe von Amazon Data Firehose
- Daten aus Kinesis Data Streams mithilfe anderer AWS Dienste lesen
- Lesen Sie mithilfe von Integrationen von Drittanbietern aus Kinesis Data Streams
- Problembehandlung bei Kinesis Data Streams Streams-Verbrauchern
- Optimieren Sie die Verbraucher von Amazon Kinesis Data Streams

Entwickeln Sie verbesserte Fan-Out-Verbraucher mit dediziertem Durchsatz

In Amazon Kinesis Data Streams können Sie Verbraucher erstellen, die ein Feature namens erweiterte Rundsendungen verwenden. Mit dieser Funktion können Verbraucher Datensätze aus einem Stream mit einem Datendurchsatz von bis zu 2 MB pro Sekunde und Shard empfangen. Dieser Durchsatz ist dediziert, d. h. dass Verbraucher, die ein erweitertes Rundsenden verwenden, nicht mit anderen Verbrauchern konkurrieren müssen, die Daten aus dem Stream empfangen. Kinesis Data Streams überträgt Datensätze aus dem Stream zu den Verbrauchern die ein erweitertes Rundsenden verwenden. Aus diesem Grund müssen diese Verbraucher keine Daten abfragen.

> **Important**
>
> Im On-Demand-Advantage-Modus können Sie bis zu 50 Verbraucher pro Stream registrieren, um den erweiterten Fan-Out zu nutzen. Bei den On-Demand-Streams Standard und Provisioned können Sie bis zu 20 Verbraucher pro Stream registrieren, um den erweiterten Fan-Out zu nutzen.

Das folgende Diagramm zeigt die Architektur für das erweiterte Rundsenden. Wenn Sie die Version 2.0 oder höher der Amazon Kinesis Client Library (KCL) verwenden, um einen Verbraucher zu erstellen, richtet die KCL den Verbraucher so ein, dass er erweitertes Rundsenden verwendet, um Daten von allen Shards des Streams zu empfangen. Wenn Sie die API verwenden, um einen Verbraucher zu erstellen, der ein erweitertes Rundsenden verwendet, können Sie einzelne Shards abonnieren.



Das Diagramm zeigt Folgendes:

- Einen Stream mit zwei Shards.
- Zwei Verbraucher, die erweitertes Rundsenden verwenden, um Daten vom Stream zu empfangen: Consumer X und Consumer Y. Beide Verbraucher haben alle Shards und alle Datensätze im Stream abonniert. Wenn Sie die Version 2.0 oder höher der KCL verwenden, um einen Verbraucher zu erstellen, abonniert die KCL automatisch alle Shards des Streams. Wenn Sie dagegen die API verwenden, um einen Verbraucher zu erstellen, können Sie einzelne Shards abonnieren.
- Pfeile stellen die Pipes für das erweiterte Rundsenden dar, die die Verbraucher verwenden, um Daten aus dem Stream zu erhalten. Eine erweiterte Fanout-Pipe stellt bis zu 2 MB/sec Daten pro Shard bereit, unabhängig von anderen Pipes oder der Gesamtzahl der Verbraucher.

Themen

- Unterschiede zwischen einem Verbraucher mit gemeinsamem Durchsatz und einem Verbraucher mit erweitertem Fan-Out-Anschluss
- Unterstützte Regionen für bis zu 50 erweiterte Fan-Out-Nutzer (nur On-Demand-Vorteil)
- Verwalten Sie Nutzer mit erweitertem Fan-Out-Status mit der Option oder AWS CLI APIs

Unterschiede zwischen einem Verbraucher mit gemeinsamem Durchsatz und einem Verbraucher mit erweitertem Fan-Out-Anschluss

In der folgenden Tabelle werden Standardverbraucher mit gemeinsamem Durchsatz und Verbrauchern mit erweitertem Fan-Out-Wert verglichen. Die Verzögerung der Nachrichtenweiterleitung ist definiert als die Zeit in Millisekunden, die eine Nutzlast, die mithilfe des Payload-Dispatchings APIs (like `PutRecord` und`PutRecords`) gesendet wurde, bis sie die Verbraucheranwendung über die nutzlastverbrauchende Anwendung (like und) erreicht. APIs `GetRecords` `SubscribeToShard`

In dieser Tabelle werden Verbraucher mit gemeinsamem Durchsatz und Verbrauchern mit erweitertem Fan-Out-Wert verglichen

Merkmale	Verbraucher mit gemeinsam em Durchsatz ohne erweitert en Fan-Out-Wert	Verbraucher mit verbesserter Fan-Out-Funktionalität
Lesedurchsatz	Der Wert wurde auf insgesamt 2 MB/sec pro Shard festgeleg t. Wenn mehrere Konsument en aus demselben Shard lesen, teilen sie sich diesen Durchsatz. Die Summe des Lesedurchsatzes der einzelnen Konsumenten eines Shards beträgt maximal 2 MB/ s.	Der Wert erhöht sich nur, wenn Konsumenten für erweiterte Rundsendungen registriert werden. Jeder für erweiterte Rundsendungen registrierte Konsument erhält einen eigenen Lesedurchsatz von bis zu 2 MB/s pro Shard, den er nicht mit anderen Konsumenten teilen muss.
Verzögerung der Nachricht enverbreitung	Ein Durchschnitt von etwa 200 ms, wenn Sie einen Verbrauch er haben, der aus dem Stream liest. Dieser Durchschnitt steigt bis zu 1000 ms an, wenn Sie fünf Verbraucher haben.	In der Regel durchschnittlich 70 ms, unabhängig davon, ob Sie einen oder fünf Verbrauch er haben.
Cost (Kosten)	Nicht zutreffend	Es gibt Datenabrufkosten und Konsumenten-Shard-

Merkmale	Verbraucher mit gemeinsamem Durchsatz ohne erweiterten Fan-Out-Wert	Verbraucher mit verbesserter Fan-Out-Funktionalität
		Stundensätze. Weitere Informationen finden Sie unter Preise für Amazon Kinesis Daten-Streams.
Datensatzbereitstellungsmodell	Modell über HTTP abrufen mit GetRecords.	Kinesis Data Streams überträgt die Datensätze über HTTP/2 an Sie. Subscribe ToShard

Unterstützte Regionen für bis zu 50 erweiterte Fan-Out-Nutzer (nur On-Demand-Vorteil)

Support für bis zu 50 erweiterte Fanout-Kunden im On-Demand-Advantage-Modus ist nur in den folgenden AWS Regionen verfügbar:

AWS Region	Name der Region
eu-north-1	Europa (Stockholm)
me-south-1	Middle East (Bahrain)
ap-south-1	Asien-Pazifik (Mumbai)
eu-west-3	Europa (Paris)
ap-southeast-3	Asien-Pazifik (Jakarta)
us-east-2	USA Ost (Ohio)
af-south-1	Afrika (Kapstadt)
eu-west-1	Europa (Irland)

AWS Region	Name der Region
me-central-1	Naher Osten (VAE)
eu-central-1	Europa (Frankfurt)
sa-east-1	Südamerika (São Paulo)
ap-east-1	Asien-Pazifik (Hongkong)
ap-south-2	Asien-Pazifik (Hyderabad)
us-east-1	USA Ost (Nord-Virginia)
ap-northeast-2	Asien-Pazifik (Seoul)
ap-northeast-3	Asien-Pazifik (Osaka)
eu-west-2	Europa (London)
ap-southeast-4	Asien-Pazifik (Melbourne)
ap-northeast-1	Asien-Pazifik (Tokio)
us-west-2	USA West (Oregon)
us-west-1	USA West (Nordkalifornien)
ap-southeast-1	Asien-Pazifik (Singapur)
ap-southeast-2	Asien-Pazifik (Sydney)
il-central-1	Israel (Tel Aviv)
ca-central-1	Kanada (Zentral)
ca-west-1	Kanada West (Calgary)
eu-south-2	Europa (Spain)
cn-northwest-1	China (Ningxia)

AWS Region	Name der Region
eu-central-2	Europa (Zürich)
us-gov-east-1	AWS GovCloud (US-Ost)
us-gov-west-1	AWS GovCloud (US-West)

Verwalten Sie Nutzer mit erweitertem Fan-Out-Status mit der Option oder AWS CLI APIs

Verbraucher, die ein erweitertes Rundsenden in Amazon Kinesis Data Streams verwenden, können Datensätze aus einem Datenstrom mit einem dedizierten Durchsatz von bis zu 2 MB Daten pro Sekunde pro Shard empfangen. Weitere Informationen finden Sie unter Entwickeln Sie verbesserte Fan-Out-Verbraucher mit dediziertem Durchsatz.

Sie können AWS CLI oder Kinesis Data Streams verwenden, um einen Verbraucher APIs zu registrieren, zu beschreiben, aufzulisten und die Registrierung aufzuheben, der das erweiterte Fan-Out in Kinesis Data Streams verwendet.

Verwalten Sie Verbraucher mit dem AWS CLI

Mit dem können Sie erweiterte Fan-Out-Nutzer registrieren, beschreiben, auflisten und deren Registrierung aufheben. AWS CLI Beispiele finden Sie in der folgenden Dokumentation.

register-stream-consumer

Registriert einen Verbraucher für einen Kinesis-Datenstream. Sie können bei der Registrierung des Verbrauchers Tags anwenden.

describe-stream-consumer

Ruft die Beschreibung eines registrierten Verbrauchers mit einer Kombination aus Verbraucher-ARN oder einer Kombination aus Verbrauchername und Stream-ARN ab.

list-stream-consumers

Führt die Verbraucher auf, die registriert sind, um Daten aus einem Stream mithilfe eines erweiterten Fan-Outs zu empfangen.

deregister-stream-consumer

Melden Sie einen Verbraucher entweder mit einem Verbraucher-ARN oder einer Kombination aus Verbrauchername und Stream-ARN ab.

Verbraucher mithilfe der Kinesis Data Streams verwalten APIs

Mithilfe der Kinesis Data Streams können Sie erweiterte Fan-Out-Verbraucher registrieren, beschreiben, auflisten und deren Registrierung aufheben. APIs Beispiele finden Sie in der folgenden Dokumentation.

RegisterStreamConsumer

Registriert einen Verbraucher für einen Kinesis-Datenstrom mit Tags. Sie können Tags bei der Registrierung des Verbrauchers anwenden.

DescribeStreamConsumer

Ruft die Beschreibung eines registrierten Verbrauchers mit einer Kombination aus Verbraucher-ARN oder einer Kombination aus Verbrauchername und Stream-ARN ab.

ListStreamConsumers

Führt die Verbraucher auf, die registriert sind, um Daten aus einem Stream mithilfe eines erweiterten Fan-Outs zu empfangen.

DeregisterStreamConsumer

Melden Sie einen Verbraucher entweder mit einem Verbraucher-ARN oder einer Kombination aus Verbrauchername und Stream-ARN ab.

Verbraucher kennzeichnen

Sie können Streams und erweiterten Fan-Out-Consumern, die Sie in Kinesis Data Streams erstellen, Ihre eigenen Metadaten in Form von Tags zuweisen. Sie können Tags verwenden, um die Kosten Ihrer Kunden zu kategorisieren und nachzuverfolgen. Sie können den Zugriff auf Verbraucher auch mithilfe von Tags mit attributebasierter Zugriffskontrolle (ABAC) steuern. Weitere Informationen finden Sie unter Kennzeichnen Sie Ihre Amazon Kinesis Data Streams Streams-Ressourcen.

Verwenden Sie den Data Viewer in der Kinesis-Konsole

Mit dem Data Viewer in der Kinesis Management Console können Sie Datensätze innerhalb des angegebenen Shards Ihres Datenstroms anzeigen, ohne eine Verbraucheranwendung entwickeln zu müssen. Gehen Sie folgendermaßen vor, um den Data Viewer zu verwenden:

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Kinesis-Konsole unter https://console.aws.amazon.com/kinesis.
2. Wählen Sie den aktiven Datenstrom aus, dessen Datensätze Sie mit dem Data Viewer anzeigen möchten, und wählen Sie dann die Registerkarte Data Viewer.
3. Wählen Sie auf der Registerkarte Data Viewer für den ausgewählten aktiven Datenstrom den Shard aus, dessen Datensätze Sie anzeigen möchten, wählen Sie die Startposition aus und klicken Sie dann auf Datensätze abrufen. Sie können die Startposition auf einen der folgenden Werte stellen:

   - Bei Sequenznummer: Zeigt Datensätze von der Position an, die durch die im Feld Sequenznummer angegebene Sequenznummer gekennzeichnet ist.
   - Nach Sequenznummer: Zeigt Datensätze direkt nach der Position an, die durch die im Feld Sequenznummer angegebene Sequenznummer gekennzeichnet ist.
   - Bei Zeitstempel: Zeigt Datensätze von der Position an, die durch den im Zeitstempelfeld angegebenen Zeitstempel gekennzeichnet ist.
   - Horizont kürzen: Zeigt Datensätze am letzten nicht getrimmten Datensatz im Shard an, der der älteste Datensatz im Shard ist.
   - Aktuell: Zeigt Datensätze direkt nach dem neuesten Datensatz im Shard an, sodass Sie immer die neuesten Daten im Shard lesen.

     Die generierten Datensätze, die mit der angegebenen Shard-ID und Startposition übereinstimmen, werden dann in einer Datensatztabelle in der Konsole angezeigt. Es werden maximal 50 Datensätze gleichzeitig angezeigt. Um die nächste Gruppe von Datensätzen anzuzeigen, klicken Sie auf die Schaltfläche Weiter.
4. Klicken Sie auf einen einzelnen Datensatz, um die Nutzdaten dieses Datensatzes im Rohdaten- oder JSON-Format in einem separaten Fenster anzuzeigen.

Beachten Sie, dass, wenn Sie im Data Viewer auf die Schaltflächen „Datensätze abrufen“ oder „Weiter“ klicken, dadurch die GetRecordsAPI aufgerufen wird. Dies gilt auch für das GetRecordsAPI-Limit von 5 Transaktionen pro Sekunde.

Fragen Sie Ihre Datenströme in der Kinesis-Konsole ab

Auf der Registerkarte Data Analytics in der Kinesis Data Streams Console können Sie Ihre Datenströme mit SQL abfragen. Gehen Sie folgendermaßen vor, um diese Funktion zu nutzen:

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Kinesis-Konsole unter https://console.aws.amazon.com/kinesis.
2. Wählen Sie den aktiven Datenstream aus, den Sie mit SQL abfragen möchten, und wählen Sie dann die Registerkarte Datenanalyse.
3. Auf der Registerkarte Datenanalyse können Sie Datenströme mit einem Managed Apache Flink Studio-Notebook überprüfen und visualisieren. Mit Apache Zeppelin können Sie Ad-hoc-SQL-Abfragen durchführen, um Ihren Datenstrom zu überprüfen und die Ergebnisse innerhalb von Sekunden anzuzeigen. Wählen Sie auf der Registerkarte Datenanalyse die Option Ich stimme zu und wählen Sie dann Notizbuch erstellen aus, um ein Notizbuch zu erstellen.
4. Nachdem das Notizbuch erstellt wurde, wählen Sie In Apache Zeppelin öffnen. Dadurch wird Ihr Notizbuch in einem neuen Tab geöffnet. Ein Notizbuch ist eine interaktive Oberfläche, über die Sie Ihre SQL-Abfragen einreichen können. Wählen Sie die Notiz aus, die den Namen Ihres Streams enthält.
5. Sie sehen eine Notiz mit einer `SELECT` Beispielabfrage zur Ausgabe der Daten im Stream, der bereits läuft. Auf diese Weise können Sie das Schema für Ihren Datenstream anzeigen.
6. Wählen Sie auf der Registerkarte Datenanalyse die Option Beispielabfragen anzeigen aus, wenn Sie andere Abfragen ausprobieren möchten, wie z. B. sich öffnende oder verschiebbare Fenster. Kopieren Sie die Abfrage, passen Sie sie an Ihr Datenstromschema an, und führen Sie sie dann in einem neuen Absatz in Ihrer Zeppelin-Notiz aus.

Verwenden Sie die Kinesis-Clientbibliothek

Was ist die Kinesis Client Library?

Kinesis Client Library (KCL) ist eine eigenständige Java-Softwarebibliothek, die entwickelt wurde, um den Prozess der Nutzung und Verarbeitung von Daten aus Amazon Kinesis Data Streams zu vereinfachen. KCL bewältigt viele der komplexen Aufgaben im Zusammenhang mit verteilter Datenverarbeitung, sodass sich Entwickler auf die Implementierung ihrer Geschäftslogik für die Datenverarbeitung konzentrieren können. Es verwaltet Aktivitäten wie den Lastenausgleich zwischen mehreren Workern, die Reaktion auf Ausfälle von Mitarbeitern, die Überprüfung verarbeiteter Datensätze und die Reaktion auf Änderungen der Anzahl der Shards im Stream.

KCL wird häufig aktualisiert, um neuere Versionen der zugrunde liegenden Bibliotheken, Sicherheitsverbesserungen und Bugfixes zu integrieren. Wir empfehlen Ihnen, die neueste Version von KCL zu verwenden, um bekannte Probleme zu vermeiden und von allen aktuellen Verbesserungen zu profitieren. Die neueste KCL-Version finden Sie auf KCL Github.

⚠ Important

- Wir empfehlen, die neueste KCL-Version zu verwenden, um bekannte Fehler und Probleme zu vermeiden. Wenn Sie KCL 2.6.0 oder früher verwenden, führen Sie ein Upgrade auf KCL 2.6.1 oder höher durch, um ein seltenes Problem zu vermeiden, das die Shard-Verarbeitung blockieren kann, wenn sich die Stream-Kapazität ändert.
- KCL ist eine Java-Bibliothek. Support für andere Sprachen als Java wird über einen Java-basierten Daemon namens bereitgestellt. MultiLangDaemon MultiLangDaemoninteragiert mit der KCL-Anwendung über STDIN und STDOUT. Weitere Informationen zum On finden Sie unter. MultiLangDaemon GitHub Entwickeln Sie Verbraucher mit KCL in Nicht-Java-Sprachen
- Verwenden Sie die AWS SDK für Java Versionen 2.27.19 bis 2.27.23 nicht mit KCL 3.x. Diese Versionen enthalten ein Problem, das einen Ausnahmefehler im Zusammenhang mit der DynamoDB-Nutzung von KCL verursacht. Wir empfehlen, die AWS SDK für Java Version 2.28.0 oder höher zu verwenden, um dieses Problem zu vermeiden.

Die wichtigsten Funktionen und Vorteile von KCL

Im Folgenden sind die wichtigsten Funktionen und die damit verbundenen Vorteile des KCL aufgeführt:

- Skalierbarkeit: Mit KCL können Anwendungen dynamisch skalieren, indem die Verarbeitungslast auf mehrere Worker verteilt wird. Sie können Ihre Anwendung manuell oder mit automatischer Skalierung nach innen oder außen skalieren, ohne sich Gedanken über die Lastumverteilung machen zu müssen.
- Lastenausgleich: KCL verteilt die Verarbeitungslast automatisch auf die verfügbaren Mitarbeiter, was zu einer gleichmäßigen Verteilung der Arbeit auf die Mitarbeiter führt.
- Checkpointing: KCL verwaltet das Checkpointing verarbeiteter Datensätze, sodass Anwendungen die Verarbeitung an der zuletzt erfolgreich verarbeiteten Position fortsetzen können.

- Fehlertoleranz: KCL bietet integrierte Fehlertoleranzmechanismen, die sicherstellen, dass die Datenverarbeitung auch dann fortgesetzt wird, wenn einzelne Mitarbeiter ausfallen. KCL bietet at-least-once auch die Lieferung an.
- Umgang mit Änderungen auf Stream-Ebene: KCL passt sich an Splits und Zusammenführungen von Shards an, die aufgrund von Änderungen des Datenvolumens auftreten können. Es behält die Reihenfolge bei, indem es sicherstellt, dass untergeordnete Shards erst verarbeitet werden, nachdem der übergeordnete Shard fertiggestellt und mit einem Checkpoint versehen wurde.
- Überwachung: KCL lässt sich in Amazon integrieren, um die Überwachung auf CloudWatch Verbraucherebene zu ermöglichen.
- Mehrsprachenunterstützung: KCL unterstützt Java nativ und ermöglicht mehrere Nicht-Java-Programmiersprachen. MultiLangDaemon

KCL-Konzepte

In diesem Abschnitt werden die Kernkonzepte und Interaktionen der Kinesis Client Library (KCL) erläutert. Diese Konzepte sind grundlegend für die Entwicklung und Verwaltung von KCL-Verbraucheranwendungen.

- KCL-Consumer-Anwendung — eine maßgeschneiderte Anwendung zum Lesen und Verarbeiten von Datensätzen aus Kinesis-Datenströmen mithilfe der Kinesis Client Library.
- Worker — KCL-Anwendungen für Privatanwender sind in der Regel verteilt, wobei ein oder mehrere Worker gleichzeitig ausgeführt werden. KCL koordiniert die Mitarbeiter so, dass sie Daten aus dem Stream auf verteilte Weise nutzen, und verteilt die Last gleichmäßig auf mehrere Mitarbeiter.
- Scheduler — eine Klasse auf hoher Ebene, die ein KCL-Worker verwendet, um mit der Datenverarbeitung zu beginnen. Jeder KCL-Worker hat einen Scheduler. Der Scheduler initialisiert und überwacht verschiedene Aufgaben, darunter das Synchronisieren von Shard-Informationen aus Kinesis-Datenströmen, das Verfolgen von Shard-Zuweisungen zwischen Workern und das Verarbeiten von Daten aus dem Stream auf der Grundlage der dem Worker zugewiesenen Shards. Der Scheduler kann verschiedene Konfigurationen annehmen, die sich auf das Verhalten des Schedulers auswirken, z. B. den Namen des zu verarbeitenden Streams und die Anmeldeinformationen. AWS Der Scheduler initiiert die Übertragung von Datensätzen aus dem Stream an die Datensatzprozessoren.
- Datensatzprozessor — definiert die Logik dafür, wie Ihre KCL-Consumer-Anwendung die Daten verarbeitet, die sie aus den Datenströmen empfängt. Sie müssen Ihre eigene benutzerdefinierte

Datenverarbeitungslogik im Record Processor implementieren. Ein KCL-Worker instanziiert einen Scheduler. Der Scheduler instanziiert dann einen Datensatzprozessor für jeden Shard, für den er eine Lease hält. Ein Worker kann mehrere Datensatzprozessoren ausführen.

- Lease — definiert die Zuordnung zwischen einem Worker und einem Shard. KCL-Anwendungen für Privatanwender verwenden Leases, um die Verarbeitung von Datensätzen auf mehrere Mitarbeiter zu verteilen. Jeder Shard ist jeweils nur an einen Arbeiter durch einen Leasingvertrag gebunden, und jeder Arbeiter kann einen oder mehrere Leasingverträge gleichzeitig abschließen. Wenn ein Arbeitnehmer aufgrund einer Kündigung oder eines Versagens den Mietvertrag beendet, beauftragt KCL einen anderen Mitarbeiter mit der Übernahme des Leasingvertrags. Weitere Informationen zum Leasing finden Sie in der Github-Dokumentation: Lease Lifecycle.
- Leasing-Tabelle — ist eine einzigartige Amazon DynamoDB-Tabelle, mit der alle Leasingverträge für die KCL-Verbraucheranwendung verfolgt werden. Jede KCL-Consumer-Anwendung erstellt ihre eigene Leasing-Tabelle. Die Leasing-Tabelle wird verwendet, um den Status aller Mitarbeiter aufrechtzuerhalten und die Datenverarbeitung zu koordinieren. Weitere Informationen finden Sie unter DynamoDB-Metadatentabellen und Lastenausgleich in KCL.
- Checkpointing — ist der Prozess, bei dem die Position des zuletzt erfolgreich verarbeiteten Datensatzes dauerhaft in einem Shard gespeichert wird. KCL verwaltet Checkpoints, um sicherzustellen, dass die Verarbeitung von der letzten Checkpoint-Position aus wieder aufgenommen werden kann, falls ein Worker ausfällt oder die Anwendung neu gestartet wird. Checkpoints werden in der DynamoDB-Leasetabelle als Teil der Metadaten des Leases gespeichert. Auf diese Weise können Mitarbeiter die Verarbeitung dort fortsetzen, wo der vorherige Mitarbeiter aufgehört hat.

DynamoDB-Metadatentabellen und Lastenausgleich in KCL

KCL verwaltet Metadaten wie Leasingverträge und CPU-Nutzungskennzahlen von Mitarbeitern. KCL verfolgt diese Metadaten mithilfe von DynamoDB-Tabellen. Für jede Amazon Kinesis Data Streams Streams-Anwendung erstellt KCL drei DynamoDB-Tabellen zur Verwaltung der Metadaten: Leasetabelle, Worker-Metriktabelle und Koordinatorstatustabelle.

> **Note**
>
> Mit KCL 3.x wurden zwei neue Metadatentabellen eingeführt: Worker-Metriken und Tabellen mit Koordinatorstatus.

> **Important**
>
> Sie müssen die richtigen Berechtigungen für KCL-Anwendungen hinzufügen, um Metadatentabellen in DynamoDB zu erstellen und zu verwalten. Details hierzu finden Sie unter Für KCL-Anwendungen für Privatanwender sind IAM-Berechtigungen erforderlich. Die KCL-Verbraucheranwendung entfernt diese drei DynamoDB-Metadatentabellen nicht automatisch. Stellen Sie sicher, dass Sie diese von der KCL-Consumer-Anwendung erstellten DynamoDB-Metadatentabellen entfernen, wenn Sie Ihre Consumer-Anwendung außer Betrieb nehmen, um unnötige Kosten zu vermeiden.

Tabelle leasen

Eine Leasing-Tabelle ist eine einzigartige Amazon DynamoDB-Tabelle, mit der die Shards nachverfolgt werden, die von den Schedulern der KCL-Consumer-Anwendung geleast und verarbeitet werden. Jede KCL-Consumer-Anwendung erstellt ihre eigene Leasing-Tabelle. KCL verwendet standardmäßig den Namen der Verbraucheranwendung als Namen der Leasetabelle. Sie können mithilfe der Konfiguration einen benutzerdefinierten Tabellennamen festlegen. KCL erstellt außerdem einen globalen sekundären Index für die Leasetabelle mit dem Partitionsschlüssel von LeaseOwner für eine effiziente Leasing-Erkennung. Der globale sekundäre Index spiegelt das LeaseKey-Attribut aus der Basis-Leasing-Tabelle wider. Wenn die Leasing-Tabelle für Ihre KCL-Consumer-Anwendung beim Start der Anwendung nicht vorhanden ist, erstellt einer der Worker die Leasing-Tabelle für Ihre Anwendung.

Sie können die Leasetabelle mit der Amazon-DynamoDB-Konsole anzeigen, während die Konsumentenanwendung ausgeführt wird.

> **Important**
>
> - Der Name jeder KCL-Consumer-Anwendung muss eindeutig sein, um zu verhindern, dass der Name der Leasetabelle doppelt vorkommt.
> - Ihr Konto wird neben den Kosten für Kinesis Data Streams mit den Kosten belastet, die für die DynamoDB-Tabelle anfallen.

Jede Zeile in der Leasetabelle steht für einen Shard, der von den Schedulern Ihrer Consumer-Anwendung verarbeitet wird. Zu den wichtigsten Feldern gehören die folgenden:

- LeaseKey: Für die Single-Stream-Verarbeitung ist dies die Shard-ID. Für die Multistream-Verarbeitung mit KCL ist es wie folgt strukturiert`account-id:StreamName:streamCreationTimestamp:ShardId`. leaseKey ist der Partitionsschlüssel der Leasetabelle. Weitere Hinweise zur Multistream-Verarbeitung finden Sie unter. Multistream-Verarbeitung mit KCL
- checkpoint: Die letzte Prüfpunkt-Sequenznummer des Shards.
- checkpointSubSequenceNummer: Wenn Sie die Aggregationsfunktion der Kinesis Producer Library verwenden, ist dies eine Erweiterung des Checkpoints, mit der einzelne Benutzerdatensätze innerhalb des Kinesis-Datensatzes verfolgt werden.
- LeaseCounter: Wird verwendet, um zu überprüfen, ob ein Mitarbeiter den Mietvertrag gerade aktiv bearbeitet. LeaseCounter erhöht sich, wenn der Leasingbesitz auf einen anderen Mitarbeiter übertragen wird.
- LeaseOwner: Der aktuelle Arbeitnehmer, der diesen Mietvertrag innehat.
- ownerSwitchesSinceCheckpoint: Wie oft hat dieser Mietvertrag seit dem letzten Checkpoint die Mitarbeiter gewechselt?
- parentShardId: ID des übergeordneten Elements dieses Shards. Stellt sicher, dass der übergeordnete Shard vollständig verarbeitet ist, bevor die Verarbeitung der untergeordneten Shards beginnt. Dabei wird die korrekte Reihenfolge der Datensatzverarbeitung beibehalten.
- childShardId: Liste der untergeordneten Shards, die beim IDs Teilen oder Zusammenführen dieses Shards entstanden sind. Wird verwendet, um die Herkunft des Shards nachzuverfolgen und die Verarbeitungsreihenfolge bei Resharding-Vorgängen zu verwalten.
- startingHashKey: Die Untergrenze des Hash-Schlüsselbereichs für diesen Shard.
- endingHashKey: Die Obergrenze des Hash-Schlüsselbereichs für diesen Shard.

Wenn Sie die Multistream-Verarbeitung mit KCL verwenden, werden die folgenden zwei zusätzlichen Felder in der Leasetabelle angezeigt. Weitere Informationen finden Sie unter Multistream-Verarbeitung mit KCL.

- shardID: Die ID des Shards.
- StreamName: Der Bezeichner des Datenstroms im folgenden Format:. `account-id:StreamName:streamCreationTimestamp`

Tabelle mit den Arbeitsmetriken

Die Tabelle mit den Worker-Metriken ist eine eindeutige Amazon DynamoDB-Tabelle für jede KCL-Anwendung und wird verwendet, um die CPU-Nutzungsmetriken für jeden Worker aufzuzeichnen. Diese Kennzahlen werden von KCL verwendet, um effiziente Leasingaufträge durchzuführen, um eine ausgewogene Ressourcennutzung aller Mitarbeiter zu erreichen. KCL verwendet standardmäßig `KCLApplicationName-WorkerMetricStats` für den Namen der Tabelle mit den Arbeitsmetriken.

Tabelle mit Status des Koordinators

Eine Koordinatorstatustabelle ist eine eindeutige Amazon DynamoDB-Tabelle für jede KCL-Anwendung und wird verwendet, um interne Statusinformationen für Mitarbeiter zu speichern. In der Tabelle mit dem Status des Koordinators werden beispielsweise Daten zur Wahl des Vorsitzenden oder Metadaten im Zusammenhang mit der direkten Migration von KCL 2.x zu KCL 3.x gespeichert. KCL verwendet standardmäßig `KCLApplicationName-CoordinatorState` für den Namen der Koordinatorstatustabelle.

DynamoDB-Kapazitätsmodus für von KCL erstellte Metadatentabellen

Standardmäßig erstellt die Kinesis Client Library (KCL) DynamoDB-Metadatentabellen wie Leasetabellen, Worker-Metrik-Tabellen und Koordinatorstatustabellen im On-Demand-Kapazitätsmodus. In diesem Modus wird die Lese- und Schreibkapazität automatisch an den Datenverkehr angepasst, ohne dass eine Kapazitätsplanung erforderlich ist. Es wird dringend empfohlen, den Kapazitätsmodus als On-Demand-Modus beizubehalten, um einen effizienteren Betrieb dieser Metadatentabellen zu gewährleisten.

Wenn Sie sich dafür entscheiden, die Leasetabelle in den Modus „Bereitgestellte Kapazität“ umzuschalten, befolgen Sie diese bewährten Methoden:

- Analysieren Sie Nutzungsmuster:
  - Überwachen Sie die Lese- und Schreibmuster und Nutzungen (RCU, WCU) Ihrer Anwendung mithilfe von Amazon-Metriken. CloudWatch
  - Machen Sie sich mit den Spitzen- und Durchschnittsdurchsatzanforderungen vertraut.
- Berechnen Sie die erforderliche Kapazität:
  - Schätzen Sie die Lesekapazitätseinheiten (RCUs) und die Schreibkapazitätseinheiten (WCUs) auf der Grundlage Ihrer Analyse.
  - Berücksichtigen Sie Faktoren wie die Anzahl der Shards, die Häufigkeit der Checkpoints und die Anzahl der Mitarbeiter.

- Implementieren Sie Auto Scaling:
  - Verwenden Sie DynamoDB Auto Scaling, um die bereitgestellte Kapazität automatisch anzupassen und angemessene Mindest- und Höchstkapazitätsgrenzen festzulegen.
  - DynamoDB Auto Scaling hilft zu verhindern, dass Ihre KCL-Metadatentabelle die Kapazitätsgrenze erreicht und gedrosselt wird.
- Regelmäßige Überwachung und Optimierung:
  - Überwachen Sie kontinuierlich die CloudWatch Metriken für`ThrottledRequests`.
  - Passen Sie die Kapazität an, wenn sich Ihre Arbeitslast im Laufe der Zeit ändert.

Wenn Sie `ProvisionedThroughputExceededException` in Metadaten-DynamoDB-Tabellen für Ihre KCL-Consumer-Anwendung feststellen, müssen Sie die bereitgestellte Durchsatzkapazität der DynamoDB-Tabelle erhöhen. Wenn Sie bei der ersten Erstellung Ihrer Verbraucheranwendung ein bestimmtes Niveau an Lesekapazitätseinheiten (RCU) und Schreibkapazitätseinheiten (WCU) festlegen, ist dies möglicherweise nicht ausreichend, wenn Ihre Nutzung zunimmt. Wenn Ihre KCL-Consumer-Anwendung beispielsweise häufig Checkpoints durchführt oder auf einem Stream mit vielen Shards arbeitet, benötigen Sie möglicherweise mehr Kapazitätseinheiten. Informationen zum bereitgestellten Durchsatz in DynamoDB finden Sie unter DynamoDB-Durchsatzkapazität und Aktualisierung einer Tabelle im Amazon DynamoDB Developer Guide.

Wie KCL Mitarbeitern Leasingverträge zuweist und die Auslastung verteilt

KCL sammelt und überwacht kontinuierlich Metriken zur CPU-Auslastung von Rechenhosts, auf denen die Worker laufen, um eine gleichmäßige Verteilung der Arbeitslast sicherzustellen. Diese CPU-Nutzungsmetriken werden in der Tabelle mit den Worker-Metriken in DynamoDB gespeichert. Wenn KCL feststellt, dass einige Mitarbeiter im Vergleich zu anderen eine höhere CPU-Auslastung aufweisen, werden die Leasingverträge zwischen den Mitarbeitern neu vergeben, um die Belastung stark ausgelasteter Mitarbeiter zu verringern. Ziel ist es, die Arbeitslast gleichmäßiger auf die gesamte Anwendungsflotte für Privatanwender zu verteilen und so zu verhindern, dass einzelne Mitarbeiter überlastet werden. Da KCL die CPU-Auslastung auf die gesamte Anwendungsflotte verteilt, können Sie die Kapazität Ihrer Anwendungsflotte für Privatanwender anpassen, indem Sie die richtige Anzahl von Workern wählen oder Auto Scaling verwenden, um die Rechenkapazität effizient zu verwalten und so die Kosten zu senken.

> **Important**
>
> KCL kann nur dann CPU-Nutzungsdaten von Mitarbeitern sammeln, wenn bestimmte Voraussetzungen erfüllt sind. Details hierzu finden Sie unter Voraussetzungen. Wenn KCL keine Kennzahlen zur CPU-Auslastung von Mitarbeitern sammeln kann, verwendet KCL den Durchsatz pro Mitarbeiter, um Leasingverträge zuzuweisen und die Auslastung auf die Mitarbeiter in der Flotte zu verteilen. KCL überwacht den Durchsatz, den jeder Mitarbeiter zu einem bestimmten Zeitpunkt erhält, und weist Leasingverträge neu zu, um sicherzustellen, dass jeder Mitarbeiter aus seinen zugewiesenen Leasingverträgen einen ähnlichen Gesamtdurchsatz erhält.

Entwickeln Sie Verbraucher mit KCL

Sie können die Kinesis Client Library (KCL) verwenden, um Verbraucheranwendungen zu erstellen, die Daten aus Ihren Kinesis-Datenströmen verarbeiten.

KCL ist in mehreren Sprachen verfügbar. Dieses Thema behandelt die Entwicklung von KCL-Consumer in Java- und Nicht-Java-Sprachen.

- Die Javadoc-Referenz zur Kinesis Client Library finden Sie in der Amazon Kinesis Client Library Javadoc.
- Informationen zum Herunterladen von GitHub KCL für Java finden Sie unter Amazon Kinesis Client Library for Java.
- Informationen zur KCL für Java auf Apache Maven finden Sie im KCL Maven Central Repository.

Themen

- Entwickeln Sie Verbraucher mit KCL in Java
- Entwickeln Sie Verbraucher mit KCL in Nicht-Java-Sprachen

Entwickeln Sie Verbraucher mit KCL in Java

Voraussetzungen

Bevor Sie mit der Verwendung von KCL 3.x beginnen, stellen Sie sicher, dass Sie über Folgendes verfügen:

- Java Development Kit (JDK) 8 oder höher
- AWS SDK für Java 2.x
- Maven oder Gradle für das Abhängigkeitsmanagement

KCL sammelt Metriken zur CPU-Auslastung, wie z. B. die CPU-Auslastung, von dem Rechenhost, auf dem die Worker arbeiten, um die Last so zu verteilen, dass eine gleichmäßige Ressourcenauslastung für alle Mitarbeiter erreicht wird. Damit KCL die CPU-Nutzungsmetriken von Workern erfassen kann, müssen Sie die folgenden Voraussetzungen erfüllen:

Amazon Elastic Compute Cloud(Amazon EC2)

- Ihr Betriebssystem muss Linux OS sein.
- Sie müssen es IMDSv2in Ihrer EC2-Instance aktivieren.

Amazon Elastic Container Service (Amazon ECS) auf Amazon EC2

- Ihr Betriebssystem muss Linux OS sein.
- Sie müssen den ECS-Endpunkt für Task-Metadaten, Version 4, aktivieren.
- Ihre Amazon ECS-Container-Agent-Version muss 1.39.0 oder höher sein.

Amazon ECS auf AWS Fargate

- Sie müssen den Fargate-Task-Metadaten-Endpunkt Version 4 aktivieren. Wenn Sie die Fargate-Plattformversion 1.4.0 oder höher verwenden, ist dies standardmäßig aktiviert.
- Fargate-Plattformversion 1.4.0 oder höher.

Amazon Elastic Kubernetes Service (Amazon EKS) auf Amazon EC2

- Ihr Betriebssystem muss Linux OS sein.

Amazon EKS auf AWS Fargate

- Fargate-Plattform 1.3.0 oder höher.

> **Important**
>
> Wenn KCL keine Kennzahlen zur CPU-Auslastung von Mitarbeitern sammeln kann, greift KCL auf den Durchsatz pro Mitarbeiter zurück, um Leasingverträge zuzuweisen und die Auslastung auf die Mitarbeiter in der Flotte zu verteilen. Weitere Informationen finden Sie unter Wie KCL Mitarbeitern Leasingverträge zuweist und die Auslastung verteilt.

Installieren und fügen Sie Abhängigkeiten hinzu

Wenn Sie Maven verwenden, fügen Sie Ihrer `pom.xml` Datei die folgende Abhängigkeit hinzu. Stellen Sie sicher, dass Sie 3.x.x durch die neueste KCL-Version ersetzt haben.

```
<dependency>
    <groupId>software.amazon.kinesis</groupId>
    <artifactId>amazon-kinesis-client</artifactId>
    <version>3.x.x</version> <!-- Use the latest version -->
</dependency>
```

Wenn Sie Gradle verwenden, fügen Sie Ihrer Datei Folgendes hinzu. `build.gradle` Stellen Sie sicher, dass Sie 3.x.x durch die neueste KCL-Version ersetzt haben.

```
implementation 'software.amazon.kinesis:amazon-kinesis-client:3.x.x'
```

Sie können im Maven Central Repository nach der neuesten Version der KCL suchen.

Implementieren Sie den Verbraucher

Eine KCL-Verbraucheranwendung besteht aus den folgenden Schlüsselkomponenten:

Zentrale Komponenten

- RecordProcessor
- RecordProcessorFactory
- Scheduler
- Hauptanwendung für Privatanwender

RecordProcessor

RecordProcessor ist die Kernkomponente, in der sich Ihre Geschäftslogik für die Verarbeitung von Kinesis-Datenstream-Datensätzen befindet. Es definiert, wie Ihre Anwendung die Daten verarbeitet, die sie vom Kinesis-Stream empfängt.

Wichtigste Aufgaben:

- Initialisieren Sie die Verarbeitung für einen Shard
- Batches von Datensätzen aus dem Kinesis-Stream verarbeiten
- Die Verarbeitung für einen Shard herunterfahren (z. B. wenn der Shard geteilt oder zusammengeführt wird oder der Lease an einen anderen Host übergeben wird)
- Verwalte Checkpoints, um den Fortschritt zu verfolgen

Im Folgenden wird ein Implementierungsbeispiel gezeigt:

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.slf4j.MDC;
import software.amazon.kinesis.exceptions.InvalidStateException;
import software.amazon.kinesis.exceptions.ShutdownException;
import software.amazon.kinesis.lifecycle.events.*;
import software.amazon.kinesis.processor.ShardRecordProcessor;

public class SampleRecordProcessor implements ShardRecordProcessor {
    private static final String SHARD_ID_MDC_KEY = "ShardId";
    private static final Logger log =
 LoggerFactory.getLogger(SampleRecordProcessor.class);
    private String shardId;

    @Override
    public void initialize(InitializationInput initializationInput) {
        shardId = initializationInput.shardId();
        MDC.put(SHARD_ID_MDC_KEY, shardId);
        try {
            log.info("Initializing @ Sequence: {}",
 initializationInput.extendedSequenceNumber());
        } finally {
            MDC.remove(SHARD_ID_MDC_KEY);
        }
    }
```

```
    @Override
    public void processRecords(ProcessRecordsInput processRecordsInput) {
        MDC.put(SHARD_ID_MDC_KEY, shardId);
        try {
            log.info("Processing {} record(s)", processRecordsInput.records().size());
            processRecordsInput.records().forEach(r ->
                log.info("Processing record pk: {} -- Seq: {}", r.partitionKey(),
r.sequenceNumber())
            );

            // Checkpoint periodically
            processRecordsInput.checkpointer().checkpoint();
        } catch (Throwable t) {
            log.error("Caught throwable while processing records. Aborting.", t);
        } finally {
            MDC.remove(SHARD_ID_MDC_KEY);
        }
    }

    @Override
    public void leaseLost(LeaseLostInput leaseLostInput) {
        MDC.put(SHARD_ID_MDC_KEY, shardId);
        try {
            log.info("Lost lease, so terminating.");
        } finally {
            MDC.remove(SHARD_ID_MDC_KEY);
        }
    }

    @Override
    public void shardEnded(ShardEndedInput shardEndedInput) {
        MDC.put(SHARD_ID_MDC_KEY, shardId);
        try {
            log.info("Reached shard end checkpointing.");
            shardEndedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            log.error("Exception while checkpointing at shard end. Giving up.", e);
        } finally {
            MDC.remove(SHARD_ID_MDC_KEY);
        }
    }

    @Override
```

```
    public void shutdownRequested(ShutdownRequestedInput shutdownRequestedInput) {
        MDC.put(SHARD_ID_MDC_KEY, shardId);
        try {
            log.info("Scheduler is shutting down, checkpointing.");
            shutdownRequestedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            log.error("Exception while checkpointing at requested shutdown. Giving
 up.", e);
        } finally {
            MDC.remove(SHARD_ID_MDC_KEY);
        }
    }
}
```

Im Folgenden finden Sie eine detaillierte Erläuterung der einzelnen in diesem Beispiel verwendeten Methoden:

initialisieren (InitializationInputInitializationInput)

- Zweck: Richten Sie alle erforderlichen Ressourcen oder den Status für die Verarbeitung von Datensätzen ein.
- Wann es aufgerufen wird: Einmal, wenn KCL diesem Datensatzprozessor einen Shard zuweist.
- Die wichtigsten Punkte:
  - `initializationInput.shardId()`: Die ID des Shards, den dieser Prozessor verarbeiten wird.
  - `initializationInput.extendedSequenceNumber()`: Die Sequenznummer, von der aus die Verarbeitung gestartet werden soll.

processRecords () ProcessRecordsInput processRecordsInput

- Zweck: Verarbeitet die eingehenden Datensätze und überprüft optional den Fortschritt.
- Wann es aufgerufen wird: Wiederholt, solange der Datensatzprozessor den Leasingvertrag für den Shard hält.
- Die wichtigsten Punkte:
  - `processRecordsInput.records()`: Liste der zu verarbeitenden Datensätze.
  - `processRecordsInput.checkpointer()`: Wird verwendet, um den Fortschritt zu überprüfen.

- Stellen Sie sicher, dass Sie alle Ausnahmen während der Verarbeitung behandelt haben, um zu verhindern, dass KCL fehlschlägt.
- Diese Methode sollte idempotent sein, da derselbe Datensatz in einigen Szenarien mehrfach verarbeitet werden kann, z. B. bei Daten, die vor einem unerwarteten Absturz oder Neustart des Workers nicht überprüft wurden.
- Leeren Sie vor dem Checkpoint stets alle gepufferten Daten, um die Datenkonsistenz sicherzustellen.

LeaseLost () LeaseLostInput leaseLostInput

- Zweck: Bereinigen Sie alle Ressourcen, die für die Verarbeitung dieses Shards spezifisch sind.
- Wann es aufgerufen wird: Wenn ein anderer Scheduler den Lease für diesen Shard übernimmt.
- Die wichtigsten Punkte:
  - Checkpointing ist bei dieser Methode nicht erlaubt.

ShardEnded () ShardEndedInput shardEndedInput

- Zweck: Beenden Sie die Verarbeitung für diesen Shard und diesen Checkpoint.
- Zeitpunkt des Aufrufs: Wenn der Shard geteilt oder zusammengeführt wird, was bedeutet, dass alle Daten für diesen Shard verarbeitet wurden.
- Die wichtigsten Punkte:
  - `shardEndedInput.checkpointer()`: Wird verwendet, um das letzte Checkpointing durchzuführen.
  - Checkpointing ist bei dieser Methode zwingend erforderlich, um die Verarbeitung abzuschließen.
  - Wenn die Daten und der Checkpoint hier nicht geleert werden, kann dies zu Datenverlust oder doppelter Verarbeitung führen, wenn der Shard erneut geöffnet wird.

ShutdownRequestedInput shutdownRequestedInputshutdownRequested ()

- Zweck: Checkpoint und Bereinigen von Ressourcen, wenn KCL heruntergefahren wird.
- Wann es aufgerufen wird: Wenn KCL heruntergefahren wird (z. B. wenn die Anwendung beendet wird).
- Die wichtigsten Punkte:

- `shutdownRequestedInput.checkpointer()`: Wird verwendet, um Checkpoints vor dem Herunterfahren durchzuführen.
- Stellen Sie sicher, dass Sie Checkpointing in der Methode implementiert haben, damit der Fortschritt gespeichert wird, bevor die Anwendung beendet wird.
- Wenn die Daten und der Checkpoint hier nicht geleert werden, kann dies zu Datenverlust oder zur erneuten Verarbeitung von Datensätzen führen, wenn die Anwendung neu gestartet wird.

> **Important**
>
> KCL 3.x sorgt dafür, dass weniger Daten erneut verarbeitet werden, wenn der Mietvertrag von einem Mitarbeiter an einen anderen übergeben wird, indem ein Checkpoint ausgeführt wird, bevor der vorherige Mitarbeiter geschlossen wird. Wenn Sie die Checkpoint-Logik nicht in der `shutdownRequested()` Methode implementieren, werden Sie diesen Vorteil nicht sehen. Stellen Sie sicher, dass Sie in der Methode eine Checkpoint-Logik implementiert haben. `shutdownRequested()`

RecordProcessorFactory

RecordProcessorFactory ist verantwortlich für die Erstellung neuer RecordProcessor Instanzen. KCL verwendet diese Factory, um RecordProcessor für jeden Shard, den die Anwendung verarbeiten muss, eine neue zu erstellen.

Wichtigste Aufgaben:

- Erstellen Sie bei Bedarf neue RecordProcessor Instanzen
- Stellen Sie sicher, dass jede ordnungsgemäß initialisiert RecordProcessor ist

Im Folgenden finden Sie ein Implementierungsbeispiel:

```
import software.amazon.kinesis.processor.ShardRecordProcessor;
import software.amazon.kinesis.processor.ShardRecordProcessorFactory;

public class SampleRecordProcessorFactory implements ShardRecordProcessorFactory {
    @Override
    public ShardRecordProcessor shardRecordProcessor() {
        return new SampleRecordProcessor();
    }
```

```
}
```

In diesem Beispiel erstellt die Factory bei SampleRecordProcessor jedem Aufruf von shardRecordProcessor () ein neues. Sie können dies um jede erforderliche Initialisierungslogik erweitern.

Scheduler

Der Scheduler ist eine Komponente auf hoher Ebene, die alle Aktivitäten der KCL-Anwendung koordiniert. Es ist für die gesamte Orchestrierung der Datenverarbeitung verantwortlich.

Wichtigste Aufgaben:

- Managen Sie den Lebenszyklus von RecordProcessors
- Erledigen Sie das Leasingmanagement für Shards
- Koordinieren Sie die Checkpoints
- Verteilen Sie die Shard-Verarbeitungslast auf mehrere Worker Ihrer Anwendung
- Bewältigen Sie Signale für ein ordnungsgemäßes Herunterfahren und Beenden von Anwendungen

Der Scheduler wird normalerweise in der Hauptanwendung erstellt und gestartet. Das Implementierungsbeispiel von Scheduler finden Sie im folgenden Abschnitt, Main Consumer Application.

Hauptanwendung für Privatanwender

Die Hauptanwendung für Verbraucher verbindet alle Komponenten miteinander. Sie ist verantwortlich für die Einrichtung des KCL-Consumer, die Erstellung der erforderlichen Clients, die Konfiguration des Schedulers und die Verwaltung des Lebenszyklus der Anwendung.

Wichtigste Aufgaben:

- AWS Service-Clients einrichten (Kinesis, DynamoDB,) CloudWatch
- Konfigurieren Sie die KCL-Anwendung
- Erstellen und starten Sie den Scheduler
- Behandelt das Herunterfahren der Anwendung

Im Folgenden finden Sie ein Implementierungsbeispiel:

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchAsyncClient;
import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.kinesis.KinesisAsyncClient;
import software.amazon.kinesis.common.ConfigsBuilder;
import software.amazon.kinesis.common.KinesisClientUtil;
import software.amazon.kinesis.coordinator.Scheduler;
import java.util.UUID;

public class SampleConsumer {
    private final String streamName;
    private final Region region;
    private final KinesisAsyncClient kinesisClient;

    public SampleConsumer(String streamName, Region region) {
        this.streamName = streamName;
        this.region = region;
        this.kinesisClient =
 KinesisClientUtil.createKinesisAsyncClient(KinesisAsyncClient.builder().region(this.region));
    }

    public void run() {
        DynamoDbAsyncClient dynamoDbAsyncClient =
 DynamoDbAsyncClient.builder().region(region).build();
        CloudWatchAsyncClient cloudWatchClient =
 CloudWatchAsyncClient.builder().region(region).build();

        ConfigsBuilder configsBuilder = new ConfigsBuilder(
            streamName,
            streamName,
            kinesisClient,
            dynamoDbAsyncClient,
            cloudWatchClient,
            UUID.randomUUID().toString(),
            new SampleRecordProcessorFactory()
        );

        Scheduler scheduler = new Scheduler(
            configsBuilder.checkpointConfig(),
            configsBuilder.coordinatorConfig(),
            configsBuilder.leaseManagementConfig(),
            configsBuilder.lifecycleConfig(),
            configsBuilder.metricsConfig(),
```

```
            configsBuilder.processorConfig(),
            configsBuilder.retrievalConfig()
        );

        Thread schedulerThread = new Thread(scheduler);
        schedulerThread.setDaemon(true);
        schedulerThread.start();
    }

    public static void main(String[] args) {
        String streamName = "your-stream-name"; // replace with your stream name
        Region region = Region.US_EAST_1; // replace with your region
        new SampleConsumer(streamName, region).run();
    }
}
```

KCL erstellt standardmäßig einen Enhanced Fan-Out (EFO) -Consumer mit dediziertem Durchsatz. Weitere Informationen zu Enhanced Fan-Out finden Sie unter. Entwickeln Sie verbesserte Fan-Out-Verbraucher mit dediziertem Durchsatz Wenn Sie weniger als 2 Verbraucher haben oder keine Verzögerungen bei der Leseverteilung unter 200 ms benötigen, müssen Sie im Scheduler-Objekt die folgende Konfiguration festlegen, um Verbraucher mit gemeinsamem Durchsatz zu verwenden:

```
configsBuilder.retrievalConfig().retrievalSpecificConfig(new PollingConfig(streamName,
 kinesisClient))
```

Der folgende Code ist ein Beispiel für die Erstellung eines Scheduler-Objekts, das Verbraucher mit gemeinsamem Durchsatz verwendet:

Importe:

```
import software.amazon.kinesis.retrieval.polling.PollingConfig;
```

Kode:

```
Scheduler scheduler = new Scheduler(
            configsBuilder.checkpointConfig(),
            configsBuilder.coordinatorConfig(),
            configsBuilder.leaseManagementConfig(),
            configsBuilder.lifecycleConfig(),
            configsBuilder.metricsConfig(),
            configsBuilder.processorConfig(),
```

```
            configsBuilder.retrievalConfig().retrievalSpecificConfig(new
PollingConfig(streamName, kinesisClient))
        );/
```

Entwickeln Sie Verbraucher mit KCL in Nicht-Java-Sprachen

In diesem Abschnitt wird die Implementierung von Verbrauchern mithilfe der Kinesis Client Library (KCL) in Python, Node.js, .NET und Ruby behandelt.

KCL ist eine Java-Bibliothek. Die Support für andere Sprachen als Java wird über eine mehrsprachige Schnittstelle namens bereitgestellt. `MultiLangDaemon` Dieser Daemon basiert auf Java und wird im Hintergrund ausgeführt, wenn Sie eine KCL mit einer anderen Sprache als Java verwenden. Wenn Sie also KCL für Nicht-Java-Sprachen installieren und Ihre Consumer-App vollständig in Nicht-Java-Sprachen schreiben, muss Java aufgrund der trotzdem auf Ihrem System installiert sein. `MultiLangDaemon` Darüber hinaus `MultiLangDaemon` verfügt es über einige Standardeinstellungen, die Sie möglicherweise für Ihren Anwendungsfall anpassen müssen (z. B. die AWS-Region, mit der eine Verbindung hergestellt wird). Weitere Informationen dazu finden Sie unter MultiLangDaemon KCL-Projekt. `MultiLangDaemon` GitHub

Die Kernkonzepte bleiben zwar in allen Sprachen gleich, es gibt jedoch einige sprachspezifische Überlegungen und Implementierungen. Die Kernkonzepte zur Entwicklung von KCL-Verbrauchern finden Sie unter. Entwickeln Sie Verbraucher mit KCL in Java Ausführlichere Informationen zur Entwicklung von KCL-Consumern in Python, Node.js, .NET und Ruby sowie die neuesten Updates finden Sie in den folgenden GitHub Repositorys:

- Python: amazon-kinesis-client-python
- Node.js: amazon-kinesis-client-nodejs
- .NET: amazon-kinesis-client-net
- Rubin: amazon-kinesis-client-ruby

> **Important**
>
> Verwenden Sie nicht die folgenden Nicht-Java-Versionen der KCL-Bibliothek, wenn Sie JDK 8 verwenden. Diese Versionen enthalten eine Abhängigkeit (Logback), die mit JDK 8 nicht kompatibel ist.
>
> - KCL Python 3.0.2 und 2.2.0
> - KCL Node.js 2.3.0

- KCL.NET 3.1.0
- KCL Ruby 2.2.0

Wir empfehlen, bei der Arbeit mit JDK 8 Versionen zu verwenden, die entweder vor oder nach diesen betroffenen Versionen veröffentlicht wurden.

Multistream-Verarbeitung mit KCL

In diesem Abschnitt werden die erforderlichen Änderungen in KCL beschrieben, die es Ihnen ermöglichen, KCL-Consumer-Anwendungen zu erstellen, die mehr als einen Datenstrom gleichzeitig verarbeiten können.

Important

- Die Multistream-Verarbeitung wird nur in KCL 2.3 oder höher unterstützt.
- Die Multistream-Verarbeitung wird nicht für KCL-Benutzer unterstützt, die in anderen Sprachen als Java geschrieben sind und mit ausgeführt werden. `multilangdaemon`
- Die Multistream-Verarbeitung wird in keiner Version von KCL 1.x unterstützt.

- MultistreamTracker Schnittstelle
  - Um eine Verbraucheranwendung zu erstellen, die mehrere Streams gleichzeitig verarbeiten kann, müssen Sie eine neue Schnittstelle namens implementieren MultistreamTracker. Diese Schnittstelle enthält die `streamConfigList`-Methode, die die Liste der Datenströme und ihrer Konfigurationen zurückgibt, die von der KCL-Konsumentenanwendung verarbeitet werden sollen. Beachten Sie, dass die Datenströme, die verarbeitet werden, während der Laufzeit der Verbraucheranwendung geändert werden können. `streamConfigList`wird regelmäßig von KCL aufgerufen, um mehr über die Änderungen der zu verarbeitenden Datenströme zu erfahren.
  - Das `streamConfigList` füllt die Liste auf StreamConfig.

```
package software.amazon.kinesis.common;

import lombok.Data;
import lombok.experimental.Accessors;
```

```
@Data
@Accessors(fluent = true)
public class StreamConfig {
    private final StreamIdentifier streamIdentifier;
    private final InitialPositionInStreamExtended initialPositionInStreamExtended;
    private String consumerArn;
}
```

- Die Felder `StreamIdentifier` und `InitialPositionInStreamExtended` sind Pflichtfelder, während sie optional `consumerArn` sind. Sie müssen das `consumerArn` nur angeben, wenn Sie KCL verwenden, um eine erweiterte Fan-Out-Anwendung für Privatanwender zu implementieren.
- Weitere Informationen `StreamIdentifier` dazu finden Sie unter https://github.com/awslabs/amazon-kinesis-client/blob/v2.5.8/amazon-kinesis-client/src/main/java/software/amazon/kinesis/common/StreamIdentifier.java #L129. Um eine zu erstellen`StreamIdentifier`, empfehlen wir, eine Multistream-Instanz aus dem `streamArn` und dem zu erstellen`streamCreationEpoch`, das in KCL 2.5.0 oder höher verfügbar ist. Erstellen Sie in KCL v2.3 und v2.4, die dies nicht unterstützen`streamArm`, eine Multistream-Instanz mithilfe des folgenden Formats. `account-id:StreamName:streamCreationTimestamp` Dieses Format ist veraltet und wird ab der nächsten Hauptversion nicht mehr unterstützt.
- MultistreamTracker beinhaltet auch eine Strategie zum Löschen von Leasingverträgen alter Streams in der Leasetabelle (). formerStreamsLeases DeletionStrategy Beachten Sie, dass die Strategie während der Laufzeit der Konsumentenanwendung NICHT geändert werden kann. Weitere Informationen finden Sie unter https://github.com/awslabs/amazon-kinesis-clientb/amazon-kinesis-client/src/main/java/software/amazon/kinesis/processor/FormerStreamsLeasesDeletionStrategy/blob/0c5042dadf794fe988438436252a5a8fe70b6b0 .java.

•

Oder Sie können ConfigsBuilder mit initialisieren, `MultiStreamTracker` wenn Sie eine KCL-Consumer-Anwendung implementieren möchten, die mehrere Streams gleichzeitig verarbeitet.

```
* Constructor to initialize ConfigsBuilder with MultiStreamTracker
     * @param multiStreamTracker
     * @param applicationName
```

```
     * @param kinesisClient
     * @param dynamoDBClient
     * @param cloudWatchClient
     * @param workerIdentifier
     * @param shardRecordProcessorFactory
     */
    public ConfigsBuilder(@NonNull MultiStreamTracker multiStreamTracker, @NonNull
String applicationName,
            @NonNull KinesisAsyncClient kinesisClient, @NonNull DynamoDbAsyncClient
dynamoDBClient,
            @NonNull CloudWatchAsyncClient cloudWatchClient, @NonNull String
workerIdentifier,
            @NonNull ShardRecordProcessorFactory shardRecordProcessorFactory) {
        this.appStreamTracker = Either.left(multiStreamTracker);
        this.applicationName = applicationName;
        this.kinesisClient = kinesisClient;
        this.dynamoDBClient = dynamoDBClient;
        this.cloudWatchClient = cloudWatchClient;
        this.workerIdentifier = workerIdentifier;
        this.shardRecordProcessorFactory = shardRecordProcessorFactory;
    }
```

- Da die Multistream-Unterstützung für Ihre KCL-Consumer-Anwendung implementiert ist, enthält jede Zeile der Leasetabelle der Anwendung jetzt die Shard-ID und den Stream-Namen der mehreren Datenströme, die diese Anwendung verarbeitet.
- Wenn Multistream-Unterstützung für Ihre KCL-Verbraucheranwendung implementiert ist, hat der LeaseKey die folgende Struktur:. `account-id:StreamName:streamCreationTimestamp:ShardId` Beispiel, `111111111:multiStreamTest-1:12345:shardId-000000000336`.

> **Important**
>
> Wenn Ihre bestehende KCL-Consumer-Anwendung so konfiguriert ist, dass sie nur einen Datenstrom verarbeitet, ist der `leaseKey` (der Partitionsschlüssel für die Leasetabelle) die Shard-ID. Wenn Sie eine bestehende KCL-Consumer-Anwendung so umkonfigurieren, dass sie mehrere Datenströme verarbeitet, wird Ihre Leasing-Tabelle beschädigt, da die `leaseKey` Struktur wie folgt aussehen muss: `account-id:StreamName:StreamCreationTimestamp:ShardId` um Multistream zu unterstützen.

Verwenden Sie die AWS Glue Schemaregistrierung mit KCL

Sie können Kinesis Data Streams in die AWS Glue Schema-Registry integrieren. Mit der AWS Glue Schema-Registry können Sie Schemas zentral erkennen, steuern und weiterentwickeln und gleichzeitig sicherstellen, dass die erstellten Daten kontinuierlich anhand eines registrierten Schemas validiert werden. Ein Schema definiert die Struktur und das Format eines Datensatzes. Ein Schema ist eine versionierte Spezifikation für zuverlässige Datenveröffentlichung, -nutzung oder -speicherung. Mit der AWS Glue Schemaregistrierung können Sie die end-to-end Datenqualität und Datenverwaltung in Ihren Streaming-Anwendungen verbessern. Weitere Informationen finden Sie unter AWS Glue Schema Registry. Eine Möglichkeit, diese Integration einzurichten, ist KCL für Java.

Important

- AWS Glue Die Schemaregistrierungsintegration für Kinesis Data Streams wird nur in KCL 2.3 oder höher unterstützt.
- AWS Glue Die Schemaregistrierungsintegration für Kinesis Data Streams wird nicht für KCL-Verbraucher unterstützt, die in Nicht-Java-Sprachen geschrieben sind und mit ausgeführt werden. `multilangdaemon`
- AWS Glue Die Schemaregistrierungsintegration für Kinesis Data Streams wird in keiner Version von KCL 1.x unterstützt.

Detaillierte Anweisungen zum Einrichten der Integration von Kinesis Data Streams mit der AWS Glue Schemaregistry mithilfe von KCL finden Sie im Abschnitt „Interaktion mit Daten mithilfe der KPL/KCL Bibliotheken“ unter Anwendungsfall: Integration von Amazon Kinesis Data Streams mit der Schemaregistry. AWS Glue

Für KCL-Anwendungen für Privatanwender sind IAM-Berechtigungen erforderlich

Sie müssen der IAM-Rolle oder dem Benutzer, der Ihrer KCL-Consumer-Anwendung zugeordnet ist, die folgenden Berechtigungen hinzufügen.

Bewährte Sicherheitsmethoden für AWS schreiben die Verwendung detaillierter Berechtigungen vor, um den Zugriff auf verschiedene Ressourcen zu kontrollieren. AWS Identity and Access Management (IAM) ermöglicht die Verwaltung von Benutzern und Benutzerberechtigungen in. AWS Eine IAM-Richtlinie führt explizit zulässige Aktionen sowie Ressourcen auf, für die diese Aktionen relevant sind.

In der folgenden Tabelle sind die IAM-Mindestberechtigungen aufgeführt, die im Allgemeinen für KCL-Verbraucheranwendungen erforderlich sind:

Minimale IAM-Berechtigungen für KCL-Verbraucheranwendungen

Service	Aktionen	Ressourcen () ARNs	Zweck

Amazon Kinesis Data Streams	`DescribeStream`
	
	`DescribeStreamSummary`
	
	`RegisterStreamConsumer`
		Kinesis-Datenstream, aus dem Ihre KCL-Anwendung die Daten verarbeitet.
		
		`arn:aws:kinesis:region:account:stream/StreamName`
			Vor dem Versuch, Datensätze zu lesen, prüft der Verbraucher, ob der Daten-Stream vorhanden und ob er aktiv ist und ob Shards im Daten-Stream enthalten sind.
			
			Registriert Verbraucher auf einem Shard.

Amazon Kinesis Data Streams	`GetRecords`
	
	`GetShardIterator`
	
	`ListShards`
		Kinesis-Datenstream, aus dem Ihre KCL-Anwendung die Daten verarbeitet.
		
		`arn:aws:kinesis:region:account:stream/StreamName`
			Auslesen von Datensätzen aus einem Shard.

Amazon Kinesis Data Streams	`SubscribeToShard`
	
	`DescribeStreamConsumer`
		Kinesis-Datenstream, aus dem Ihre KCL-Anwendung die Daten verarbeitet. Fügen Sie diese Aktion nur hinzu, wenn Sie Enhanced Fanout (EFO) -Verbraucher verwenden.	Abonniert einen Shard für Enhanced Fan-Out (EFO) -Verbraucher.

Service	Aktionen	Ressourcen () ARNs	Zweck
		`arn:aws:kinesis:region:account:stream/StreamName/consumer/*`	

Amazon DynamoDB	`CreateTable`
	`DescribeTable`
	`UpdateTable`
	`Scan`
	`GetItem`
	`PutItem`
	`UpdateItem`
	`DeleteItem`
		Leasetabelle (von KCL erstellte Metadatentabelle in DynamoDB).
		`arn:aws:dynamodb:region:account:table/KCLApplicationName`
			Diese Aktionen sind erforderlich, damit KCL die in DynamoDB erstellte Leasetabelle verwalten kann.

Service	Aktionen	Ressourcen () ARNs	Zweck

Amazon DynamoDB	`CreateTable`
	
	`DescribeTable`
	
	`Scan`
	
	`GetItem`
	
	`PutItem`
	
	`UpdateItem`
	
	`DeleteItem`
		Von KCL erstellte Worker-Metriken und Koordinat orstatustabelle (Metadatentabellen in DynamoDB).
		
		`arn:aws:dynamodb:region:account:table/KCLApplicationName-WorkerMetricStats`
		
		`arn:aws:dynamodb:region:account:table/KCLApplicationName-CoordinatorState`
			Diese Aktionen sind erforderlich, damit KCL die Worker-Metriken und die Koordinatorstatus-Metadatentabellen in DynamoDB verwalten kann.

Service	Aktionen	Ressourcen () ARNs	Zweck
Amazon DynamoDB	`Query`	Globaler sekundärer Index in der Leasing-Tabelle. `arn:aws:dynamodb:region:account:table/KCLApplicationName/index/*`	Diese Aktion ist erforderlich, damit KCL den globalen sekundären Index der in DynamoDB erstellten Leasetabelle lesen kann.
Amazon CloudWatch	`PutMetricData`	*	Laden Sie Metriken hoch CloudWatch , die für die Überwachung der Anwendung nützlich sind. Das Sternchen (*) wird verwendet, weil es keine bestimmte Ressource gibt, für die die `PutMetricData` Aktion aufgerufen wird. CloudWatch

> **Note**
>
> Ersetzen Sie „Region“, „Konto“, „StreamName“ und „KCLApplicationName“ in der ARNs durch Ihre eigene AWS-Konto Nummer AWS-Region, den Kinesis-Datenstromnamen bzw. den KCL-Anwendungsnamen. KCL 3.x erstellt zwei weitere Metadatentabellen in DynamoDB. Einzelheiten zu DynamoDB-Metadatentabellen, die von KCL erstellt wurden, finden Sie unter. DynamoDB-Metadatentabellen und Lastenausgleich in KCL Wenn Sie Konfigurationen

verwenden, um die Namen der von KCL erstellten Metadatentabellen anzupassen, verwenden Sie diese angegebenen Tabellennamen anstelle des KCL-Anwendungsnamens.

Im Folgenden finden Sie ein Beispiel für ein Richtliniendokument für eine KCL-Verbraucheranwendung.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kinesis:DescribeStream",
                "kinesis:DescribeStreamSummary",
                "kinesis:RegisterStreamConsumer",
                "kinesis:GetRecords",
                "kinesis:GetShardIterator",
                "kinesis:ListShards"
            ],
            "Resource": "arn:aws:kinesis:us-
east-1:123456789012:stream/STREAM_NAME"
        },
        {
            "Effect": "Allow",
            "Action": [
                "kinesis:SubscribeToShard",
                "kinesis:DescribeStreamConsumer"
            ],
            "Resource": "arn:aws:kinesis:us-
east-1:123456789012:stream/STREAM_NAME/consumer/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:CreateTable",
                "dynamodb:DescribeTable",
                "dynamodb:UpdateTable",
                "dynamodb:GetItem",
                "dynamodb:UpdateItem",
```

```
                "dynamodb:PutItem",
                "dynamodb:DeleteItem",
                "dynamodb:Scan"
            ],
            "Resource": [
            "arn:aws:dynamodb:us-east-1:123456789012:table/KCL_APPLICATION_NAME"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:CreateTable",
                "dynamodb:DescribeTable",
                "dynamodb:GetItem",
                "dynamodb:UpdateItem",
                "dynamodb:PutItem",
                "dynamodb:DeleteItem",
                "dynamodb:Scan"
            ],
            "Resource": [
            "arn:aws:dynamodb:us-east-1:123456789012:table/KCL_APPLICATION_NAME-
WorkerMetricStats",
    "arn:aws:dynamodb:us-east-1:123456789012:table/KCL_APPLICATION_NAME-
CoordinatorState"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "dynamodb:Query"
            ],
            "Resource": [
            "arn:aws:dynamodb:us-east-1:123456789012:table/KCL_APPLICATION_NAME/
index/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricData"
            ],
            "Resource": "*"
        }
    ]
```

```
}
```

Bevor Sie diese Beispielrichtlinie verwenden, überprüfen Sie die folgenden Punkte:

- Ersetzen Sie REGION durch Ihr AWS-Region (zum Beispiel us-east-1).
- Ersetzen Sie ACCOUNT_ID durch Ihre ID. AWS-Konto
- Ersetzen Sie STREAM_NAME durch den Namen Ihres Kinesis-Datenstreams.
- Ersetzen Sie CONSUMER_NAME durch den Namen Ihres Verbrauchers, normalerweise Ihren Anwendungsnamen, wenn Sie KCL verwenden.
- Ersetzen Sie KCL_APPLICATION_NAME durch den Namen Ihrer KCL-Anwendung.

KCL-Konfigurationen

Sie können Konfigurationseigenschaften festlegen, um die Funktionalität der Kinesis Client Library an Ihre spezifischen Anforderungen anzupassen. In der folgenden Tabelle werden die Konfigurationseigenschaften und Klassen beschrieben.

> **Important**
>
> In KCL 3.x zielt der Load-Balancing-Algorithmus darauf ab, eine gleichmäßige CPU-Auslastung für alle Worker zu erreichen und nicht die gleiche Anzahl von Leases pro Worker. Wenn der `maxLeasesForWorker` Wert zu niedrig ist, schränken Sie möglicherweise die Fähigkeit von KCL ein, die Arbeitslast effektiv auszugleichen. Wenn Sie die `maxLeasesForWorker` Konfiguration verwenden, sollten Sie erwägen, ihren Wert zu erhöhen, um eine bestmögliche Lastverteilung zu ermöglichen.

Diese Tabelle zeigt die Konfigurationseigenschaften für KCL

Konfigurationseigenschaft	Konfigurationsklasse	Description	Standardwert
`applicationName`	ConfigsBuilder	Der Name für die KCL-Anwendung. Wird als Standard für `tableName` und	Nicht zutreffend

Konfigurationseigenschaft	Konfigurationsklasse	Description	Standardwert
		`consumerName` verwendet.	
`tableName`	ConfigsBuilder	Ermöglicht das Überschreiben des für die Lease-Tabelle von Amazon DynamoDB verwendeten Tabellennamens.	Nicht zutreffend
`streamName`	ConfigsBuilder	Der Name des Streams, dessen Datensätze diese Anwendung verarbeitet.	Nicht zutreffend
`workerIdentifier`	ConfigsBuilder	Eine eindeutige Kennung, die die Instanziierung des Anwendungsprozessors repräsentiert. Dieser Wert muss eindeutig sein.	Nicht zutreffend

Konfigurationseigenschaft	Konfigurationsklasse	Description	Standardwert
`failoverTimeMillis`	LeaseManagementConfig	Anzahl Millisekunden, nach deren Ablauf unterstellt werden kann, dass ein Lease-Eigentümer fehlgeschlagen ist. Für Anwendungen mit einer großen Anzahl von Shards kann diese Zahl auf eine höhere Anzahl festgelegt werden, um die Anzahl der DynamoDB-IOPS zu reduzieren, die für die Nachverfolgung von Leases erforderlich sind.	10.000 (10 Sekunden)
`shardSyncIntervalMillis`	LeaseManagementConfig	Die Zeit zwischen Shard-Synchronisierungsaufrufen.	60.000 (60 Sekunden)
`cleanupLeasesUponShardCompletion`	LeaseManagementConfig	Wenn diese Option aktiviert ist, werden Leases entfernt, sobald die untergeordneten Leases die Verarbeitung gestartet haben.	TRUE

Konfigurationseigenschaft	Konfigurationsklasse	Description	Standardwert
`ignoreUnexpectedChildShards`	LeaseManagementConfig	Wenn diese Option aktiviert ist, werden untergeordnete Shards, die einen offenen Shard aufweisen, ignoriert. Dies gilt hauptsächlich für DynamoDB Streams.	FALSE
`maxLeasesForWorker`	LeaseManagementConfig	Die maximale Anzahl von Leasingverträgen, die ein einzelner Arbeitnehmer annehmen sollte. Eine zu niedrige Einstellung kann zu Datenverlusten führen, wenn Mitarbeiter nicht alle Shards verarbeiten können, was zu einer suboptimalen Leasingzuweisung zwischen den Mitarbeitern führen kann. Berücksichtigen Sie bei der Konfiguration die Gesamtzahl der Shards, die Anzahl der Worker und die Verarbeitungskapazität der Mitarbeiter.	Unbegrenzt

Konfigurationseigenschaft	Konfigurationsklasse	Description	Standardwert
`maxLeaseRenewalThreads`	LeaseManagementConfig	Steuert die Größe des Lease-Renewer-Thread-Pools. Dieser Pool muss größer sein, wenn die Anwendung mehr Leases annehmen kann.	20
`billingMode`	LeaseManagementConfig	Bestimmt den Kapazitätsmodus der in DynamoDB erstellten Leasetabelle. Es gibt zwei Optionen: den On-Demand-Modus (PAY_PER_REQUEST) und den Bereitstellungsmodus. Wir empfehlen, die Standardeinstellung des On-Demand-Modus zu verwenden, da dieser automatisch an Ihre Arbeitslast angepasst wird, ohne dass eine Kapazitätsplanung erforderlich ist.	PAY_PER_REQUEST (On-Demand-Modus)

Konfigurationseigenschaft	Konfigurationsklasse	Description	Standardwert
`initialLeaseTableReadCapacity`	LeaseManagementConfig	Die DynamoDB-Lesekapazität, die verwendet wird, wenn die Kinesis-Clientbibliothek eine neue DynamoDB-Leasetabelle mit bereitgestelltem Kapazitätsmodus erstellen muss. Sie können diese Konfiguration ignorieren, wenn Sie in der Konfiguration den standardmäßigen On-Demand-Kapazitätsmodus verwenden. `billingMode`	10

Konfigurationseigenschaft	Konfigurationsklasse	Description	Standardwert
`initialLeaseTableWriteCapacity`	LeaseManagementConfig	Die DynamoDB-Lesekapazität, die verwendet wird, wenn die Kinesis-Clientbibliothek eine neue DynamoDB-Leasetabelle erstellen muss. Sie können diese Konfiguration ignorieren, wenn Sie in der Konfiguration den standardmäßigen On-Demand-Kapazitätsmodus verwenden. `billingMode`	10
`initialPositionInStreamExtended`	LeaseManagementConfig	Die ursprüngliche Position im Stream, an der die Anwendung starten soll. Dieser Wert wird nur im Rahmen der Lease-Erstellung verwendet.	InitialPositionInStream.TRIM_HORIZON

Konfigurationseigenschaft	Konfigurationsklasse	Description	Standardwert
`reBalanceThresholdPercentage`	LeaseManagementConfig	Ein Prozentwert, der bestimmt, wann der Load-Balancing-Algorithmus die Neuzuweisung von Shards unter Workern in Betracht ziehen sollte. Dies ist eine neue Konfiguration, die in KCL 3.x eingeführt wurde.	10
`dampeningPercentage`	LeaseManagementConfig	Ein Prozentwert, der verwendet wird, um die Menge an Last zu dämpfen, die bei einem einzigen Ausgleichsvorgang vom überlasteten Arbeiter abtransportiert wird. Dies ist eine neue Konfiguration, die in KCL 3.x eingeführt wurde.	60

Konfigurationseigenschaft	Konfigurationsklasse	Description	Standardwert
`allowThroughputOvershoot`	LeaseManagementConfig	Ermittelt, ob dem überlasteten Worker weiterhin zusätzliche Leasingverträge abgenommen werden müssen, auch wenn dadurch der gesamte Leasing-Durchsatz den gewünschten Durchsatz übersteigt. Dies ist eine neue Konfiguration, die in KCL 3.x eingeführt wurde.	TRUE

Konfigurationseigenschaft	Konfigurationsklasse	Description	Standardwert
`disableWorkerMetrics`	LeaseManagementConfig	Legt fest, ob KCL bei der Neuzuweisung von Leasingverträgen und beim Lastenausgleich Ressourcenmetriken von Workern (z. B. CPU-Auslastung) ignorieren soll. Setzen Sie diesen Wert auf TRUE, wenn Sie verhindern möchten, dass KCL einen Lastenausgleich auf der Grundlage der CPU-Auslastung durchführt. Dies ist eine neue Konfiguration, die in KCL 3.x eingeführt wurde.	FALSE

Konfigurationseigenschaft	Konfigurationsklasse	Description	Standardwert
`maxThroughputPerHostKBps`	LeaseManagementConfig	Höhe des maximalen Durchsatzes, der einem Mitarbeiter während der Leasingzuweisung zugewiesen werden soll. Dies ist eine neue Konfiguration, die in KCL 3.x eingeführt wurde.	Unbegrenzt

Konfigurationseigenschaft	Konfigurationsklasse	Description	Standardwert
`isGracefulLeaseHandoffEnabled`	LeaseManagementConfig	Steuert das Verhalten bei der Übergabe von Leasingverträgen zwischen Mitarbeitern. Wenn dieser Wert auf „true“ gesetzt ist, versucht KCL, Leasingverträge ordnungsgemäß zu übertragen, indem dem Shard RecordProcessor genügend Zeit eingeräumt wird, um die Verarbeitung abzuschließen, bevor der Leasing an einen anderen Worker übergeben wird. Dies kann dazu beitragen, die Dateninte grität und reibungslose Übergänge zu gewährleisten, kann jedoch die Übergabezeit verlängern. Wenn der Wert auf „Falsch“ gesetzt ist, wird der Leasingvertrag sofort übergeben, ohne darauf RecordProcessor zu warten, dass er ordnungsgemäß	TRUE

Konfigurationseigenschaft	Konfigurationsklasse	Description	Standardwert
		beendet wird. Dies kann zu schnelleren Übergaben führen, es besteht jedoch die Gefahr einer unvollständigen Bearbeitung. Hinweis: Checkpointing muss innerhalb der shutdownRequested () -Methode von implementiert werden, um von der Funktion „Graceful Lease Handoff“ RecordProcessor zu profitieren. Dies ist eine neue Konfiguration, die in KCL 3.x eingeführt wurde.	

Konfigurationseigenschaft	Konfigurationsklasse	Description	Standardwert
`gracefulLeaseHandoffTimeoutMillis`	LeaseManagementConfig	Gibt die Mindestzeit (in Millisekunden) an, um zu warten, bis die aktuellen Shards ordnungsgemäß heruntergefahren wurden, bevor das Leasing gewaltsam RecordProcessor an den nächsten Besitzer übertragen wird. Wenn Ihre ProcessRecords-Methode normalerweise länger als der Standardwert ausgeführt wird, sollten Sie erwägen, diese Einstellung zu erhöhen. Dadurch wird sichergestellt, dass genügend Zeit zur Verfügung RecordProcessor steht, um die Verarbeitung abzuschließen, bevor die Leasingübertragung erfolgt. Dies ist eine neue Konfiguration, die in KCL 3.x eingeführt wurde.	30.000 (30 Sekunden)

Konfigurationseigenschaft	Konfigurationsklasse	Description	Standardwert
`maxRecords`	PollingConfig	Ermöglicht das Einstellen der maximalen Anzahl an Datensätzen, die Kinesis zurückgibt.	10.000
`retryGetRecordsInSeconds`	PollingConfig	Konfiguriert die Verzögerung zwischen GetRecords Fehlversuchen.	Keine
`maxGetRecordsThreadPool`	PollingConfig	Die Thread-Pool-Größe, die für GetRecords verwendet wird.	Keine
`idleTimeBetweenReadsInMillis`	PollingConfig	Legt fest, wie lange KCL zwischen GetRecords Aufrufen wartet, um die Daten aus Datenströmen abzufragen. Die Einheit ist Millisekunden.	1.500
`callProcessRecordsEvenForEmptyRecordList`	ProcessorConfig	Wenn diese Option aktiviert ist, wird der Datensatzprozessor aufgerufen, auch wenn Kinesis keine Datensätze bereitgestellt hat.	FALSE

Konfigurationseigenschaft	Konfigurationsklasse	Description	Standardwert
`parentShardPollIntervalMillis`	CoordinatorConfig	Gibt an, wie oft ein Datensatzprozessor abfragen soll, ob der übergeordnete Shard abgeschlossen wurde. Die Einheit ist Millisekunden.	10.000 (10 Sekunden)
`skipShardSyncAtWorkerInitializationIfLeaseExist`	CoordinatorConfig	Synchronisieren der Shard-Daten deaktivieren, wenn die Lease-Tabelle Leases enthält.	FALSE
`shardPrioritization`	CoordinatorConfig	Welche Shard-Priorisierung verwendet werden soll.	NoOpShardPrioritization

Konfigurationseigenschaft	Konfigurationsklasse	Description	Standardwert
`ClientVersionConfig`	CoordinatorConfig	Legt fest, in welchem KCL-Versionskompatibilitätsmodus die Anwendung ausgeführt wird. Diese Konfiguration ist nur für die Migration von früheren KCL-Versionen vorgesehen. Bei der Migration auf 3.x müssen Sie diese Konfiguration auf einstellen. `CLIENT_VERSION_CONFIG_COMPATIBLE_WITH_2X` Sie können diese Konfiguration entfernen, wenn Sie die Migration abgeschlossen haben.	CLIENT_VERSION_CONFIG_3X
`taskBackoffTimeMillis`	LifecycleConfig	Die Wartezeit für die Wiederholung fehlgeschlagener KCL-Aufgaben. Die Einheit ist Millisekunden.	500 (0,5 Sekunden)

Konfigurationseigenschaft	Konfigurationsklasse	Description	Standardwert
`logWarningForTaskAfterMillis`	LifecycleConfig	Wartezeit, bevor eine Warnung protokolliert wird, wenn eine Aufgabe nicht abgeschlossen wurde.	Keine
`listShardsBackoffTimeInMillis`	RetrievalConfig	Die Anzahl der zwischen Aufrufen von `ListShards` abzuwartenden Millisekunden, wenn es zu Fehlern kommt. Die Einheit ist Millisekunden.	1.500 (1,5 Sekunden)
`maxListShardsRetryAttempts`	RetrievalConfig	Die maximale Anzahl Wiederholungsversuche durch `ListShards`, bevor abgebrochen wird.	50
`metricsBufferTimeMillis`	MetricsConfig	Gibt die maximale Dauer (in Millisekunden) an, für die Metriken zwischengespeichert werden sollen, bevor sie veröffentlicht werden. CloudWatch	10.000 (10 Sekunden)

Konfigurationseigenschaft	Konfigurationsklasse	Description	Standardwert
`metricsMaxQueueSize`	MetricsConfig	Gibt die maximale Anzahl von Metriken an, die vor der Veröffentlichung zwischengespeichert werden sollen CloudWatch.	10.000
`metricsLevel`	MetricsConfig	Gibt die Granularitätsstufe der CloudWatch Metriken an, die aktiviert und veröffentlicht werden sollen. Mögliche Werte: NONE, SUMMARY, DETAILLIERT.	MetricsLevel. DETAILLIERT
`metricsEnabledDimensions`	MetricsConfig	Steuert die zulässigen Dimensionen für CloudWatch Metriken.	Alle Dimensionen

Nicht mehr verfügbare Konfigurationen in KCL 3.x

Die folgenden Konfigurationseigenschaften sind in KCL 3.x nicht mehr verfügbar:

Die Tabelle zeigt die nicht mehr verfügbaren Konfigurationseigenschaften für KCL 3.x

Konfigurationseigenschaft	Konfigurationsklasse	Description
`maxLeasesToStealAtOneTime`	LeaseManagementConfig	Die maximale Anzahl der Leases, die eine Anwendung zu einem gegebenen Zeitpunkt zu stehlen

Konfigurationseigenschaft	Konfigurationsklasse	Description
		versuchen sollte. KCL 3.x ignoriert diese Konfiguration und weist Leasingverträge auf der Grundlage der Ressourcennutzung der Mitarbeiter neu zu.
`enablePriorityLeaseAssignment`	LeaseManagementConfig	Steuert, ob Mitarbeiter der Annahme sehr abgelaufener Leasingverträge (Leasingverträge, die nicht um das Dreifache der Failover-Zeit verlängert werden) und neuen Shard-Leases Priorität einräumen sollen, unabhängig von der Anzahl der angestrebten Leasingverträge, aber unter Einhaltung der maximalen Leasing-Limits. KCL 3.x ignoriert diese Konfiguration und verteilt abgelaufene Leasingverträge immer auf die Mitarbeiter.

Important

Während der Migration von früheren KCL-Versionen zu KCL 3.x müssen Sie weiterhin über die Eigenschaften der nicht mehr verfügbaren Konfiguration verfügen. Während der Migration startet der KCL-Worker zunächst mit dem KCL 2.x-kompatiblen Modus und wechselt in den KCL 3.x-Funktionsmodus, wenn er feststellt, dass alle KCL-Worker der Anwendung bereit sind, KCL 3.x auszuführen. Diese eingestellten Konfigurationen werden benötigt, solange KCL-Worker den KCL 2.x-kompatiblen Modus ausführen.

Lebenszyklus-Richtlinie für KCL-Versionen

In diesem Thema wird die Versionslebenszyklusrichtlinie für die Amazon Kinesis Client Library (KCL) beschrieben. AWS stellt regelmäßig neue Versionen für KCL-Versionen bereit, um neue Funktionen und Verbesserungen, Bugfixes, Sicherheitspatches und Abhängigkeitsupdates zu unterstützen. Wir empfehlen Ihnen, up-to-date bei den KCL-Versionen zu bleiben, um über die neuesten Funktionen, Sicherheitsupdates und die zugrunde liegenden Abhängigkeiten auf dem Laufenden zu bleiben. Wir empfehlen nicht, weiterhin eine KCL-Version zu verwenden, die nicht unterstützt wird.

Der Lebenszyklus der wichtigsten KCL-Versionen besteht aus den folgenden drei Phasen:

- Allgemeine Verfügbarkeit (GA) — Während dieser Phase wird die Hauptversion vollständig unterstützt. AWS bietet regelmäßige Versionen von Neben- und Patch-Versionen, die Unterstützung für neue Funktionen oder API-Updates für Kinesis Data Streams sowie Fehler- und Sicherheitskorrekturen beinhalten.
- Wartungsmodus — AWS Beschränkt die Veröffentlichung von Patch-Versionen auf kritische Bugfixes und Sicherheitsprobleme. Die Hauptversion erhält keine Updates für neue Funktionen oder APIs für Kinesis Data Streams.
- E nd-of-support — Die Hauptversion wird keine Updates oder Releases mehr erhalten. Zuvor veröffentlichte Versionen werden weiterhin über öffentliche Paketmanager verfügbar sein und der Code bleibt aktiviert GitHub. Die Verwendung einer Version, die erreicht wurde, end-of-support erfolgt nach eigenem Ermessen des Benutzers. Wir empfehlen Ihnen, auf die neueste Hauptversion zu aktualisieren.

Hauptversion	Aktuelle Phase	Datum der Veröffentlichung	Datum des Wartungsmodus	End-of-support Datum
KCL 1.x	Wartungsmodus	19.12.2013	2025-04-17	2026-01-30
KCL 2.x	Allgemeine Verfügbarkeit	02.08.2018	--	--
KCL 3.x	Allgemeine Verfügbarkeit	2024-11-06	--	--

Migrieren Sie von früheren KCL-Versionen

In diesem Thema wird erklärt, wie Sie von früheren Versionen der Kinesis Client Library (KCL) migrieren.

Was ist neu in KCL 3.0?

Kinesis Client Library (KCL) 3.0 bietet mehrere wichtige Verbesserungen im Vergleich zu früheren Versionen:

- Es senkt die Rechenkosten für Privatanwenderanwendungen, indem die Arbeit automatisch von überlasteten Mitarbeitern auf nicht ausgelastete Mitarbeiter in der Anwenderflotte für Privatanwender verteilt wird. Dieser neue Load-Balancing-Algorithmus sorgt für eine gleichmäßige Verteilung der CPU-Auslastung auf die Mitarbeiter und macht eine Überbelegung der Mitarbeiter überflüssig.
- Es reduziert die mit KCL verbundenen DynamoDB-Kosten, indem Lesevorgänge in der Leasetabelle optimiert werden.
- Es minimiert die erneute Verarbeitung von Daten, wenn Leasingverträge einem anderen Mitarbeiter zugewiesen werden, indem der aktuelle Mitarbeiter die Überprüfung der von ihm verarbeiteten Datensätze durchführen kann.
- Es verwendet AWS SDK for Java 2.x verbesserte Leistungs- und Sicherheitsfunktionen, wodurch die Abhängigkeit von 1.x vollständig beseitigt wird. AWS SDK für Java

Weitere Informationen finden Sie in den Versionshinweisen zu KCL 3.0.

Themen

- Migrieren Sie von KCL 2.x zu KCL 3.x
- Rollback zu einer früheren KCL-Version
- Rollforward zu KCL 3.x nach einem Rollback
- Bewährte Methoden für die Leasetabelle mit dem Modus „Bereitgestellte Kapazität“
- Migrieren von KCL 1.x zu KCL 3.x

Migrieren Sie von KCL 2.x zu KCL 3.x

Dieses Thema enthält step-by-step Anweisungen zur Migration Ihres Verbrauchers von KCL 2.x zu KCL 3.x. KCL 3.x unterstützt die direkte Migration von KCL 2.x-Verbrauchern. Sie können

weiterhin die Daten aus Ihrem Kinesis-Datenstrom nutzen und gleichzeitig Ihre Mitarbeiter fortlaufend migrieren.

> **Important**
>
> KCL 3.x verwendet dieselben Schnittstellen und Methoden wie KCL 2.x. Daher müssen Sie Ihren Datensatzverarbeitungscode während der Migration nicht aktualisieren. Sie müssen jedoch die richtige Konfiguration festlegen und die erforderlichen Schritte für die Migration überprüfen. Wir empfehlen Ihnen dringend, die folgenden Migrationsschritte zu befolgen, um eine reibungslose Migration zu gewährleisten.

Schritt 1: Voraussetzungen

Bevor Sie mit der Verwendung von KCL 3.x beginnen, stellen Sie sicher, dass Sie über Folgendes verfügen:

- Java Development Kit (JDK) 8 oder höher
- AWS SDK für Java 2.x
- Maven oder Gradle für das Abhängigkeitsmanagement

> **Important**
>
> Verwenden Sie nicht die AWS SDK für Java Versionen 2.27.19 bis 2.27.23 mit KCL 3.x. Diese Versionen enthalten ein Problem, das einen Ausnahmefehler im Zusammenhang mit der DynamoDB-Nutzung von KCL verursacht. Wir empfehlen, die AWS SDK für Java Version 2.28.0 oder höher zu verwenden, um dieses Problem zu vermeiden.

Schritt 2: Abhängigkeiten hinzufügen

Wenn Sie Maven verwenden, fügen Sie Ihrer `pom.xml` Datei die folgende Abhängigkeit hinzu. Stellen Sie sicher, dass Sie 3.x.x durch die neueste KCL-Version ersetzt haben.

```
<dependency>
    <groupId>software.amazon.kinesis</groupId>
    <artifactId>amazon-kinesis-client</artifactId>
    <version>3.x.x</version> <!-- Use the latest version -->
```

```
</dependency>
```

Wenn Sie Gradle verwenden, fügen Sie Ihrer Datei Folgendes hinzu. `build.gradle` Stellen Sie sicher, dass Sie 3.x.x durch die neueste KCL-Version ersetzt haben.

```
implementation 'software.amazon.kinesis:amazon-kinesis-client:3.x.x'
```

Sie können im Maven Central Repository nach der neuesten Version der KCL suchen.

Schritt 3: Richten Sie die migrationsbezogene Konfiguration ein

Um von KCL 2.x zu KCL 3.x zu migrieren, müssen Sie den folgenden Konfigurationsparameter festlegen:

- CoordinatorConfig. clientVersionConfig: Diese Konfiguration bestimmt, in welchem KCL-Versionskompatibilitätsmodus die Anwendung ausgeführt wird. Bei der Migration von KCL 2.x auf 3.x müssen Sie diese Konfiguration auf einstellen. `CLIENT_VERSION_CONFIG_COMPATIBLE_WITH_2X` Um diese Konfiguration festzulegen, fügen Sie bei der Erstellung Ihres Scheduler-Objekts die folgende Zeile hinzu:

```
configsBuilder.coordiantorConfig().clientVersionConfig(ClientVersionConfig.CLIENT_VERSION_CONFI
```

Im Folgenden finden Sie ein Beispiel dafür, wie Sie den `CoordinatorConfig.clientVersionConfig` für die Migration von KCL 2.x auf 3.x festlegen. Sie können andere Konfigurationen nach Bedarf an Ihre spezifischen Anforderungen anpassen:

```
Scheduler scheduler = new Scheduler(
    configsBuilder.checkpointConfig(),

 configsBuilder.coordiantorConfig().clientVersionConfig(ClientVersionConfig.CLIENT_VERSION_CONF
    configsBuilder.leaseManagementConfig(),
    configsBuilder.lifecycleConfig(),
    configsBuilder.metricsConfig(),
    configsBuilder.processorConfig(),
    configsBuilder.retrievalConfig()
);
```

Es ist wichtig, dass alle Worker in Ihrer Consumer-Anwendung zu einem bestimmten Zeitpunkt denselben Load-Balancing-Algorithmus verwenden, da KCL 2.x und 3.x unterschiedliche Load-

Balancing-Algorithmen verwenden. Das Ausführen von Workern mit unterschiedlichen Load-Balancing-Algorithmen kann zu einer suboptimalen Lastverteilung führen, da die beiden Algorithmen unabhängig voneinander arbeiten.

Diese KCL 2.x-Kompatibilitätseinstellung ermöglicht es Ihrer KCL 3.x-Anwendung, in einem mit KCL 2.x kompatiblen Modus zu laufen und den Load-Balancing-Algorithmus für KCL 2.x zu verwenden, bis alle Worker in Ihrer Consumer-Anwendung auf KCL 3.x aktualisiert wurden. Wenn die Migration abgeschlossen ist, wechselt KCL automatisch in den vollen KCL 3.x-Funktionsmodus und beginnt mit der Verwendung eines neuen KCL 3.x-Lastenausgleichsalgorhythmus für alle laufenden Worker.

> **Important**
>
> Wenn Sie Konfigurationen nicht verwenden, `ConfigsBuilder` sondern ein `LeaseManagementConfig` Objekt erstellen, müssen Sie einen weiteren Parameter hinzufügen, der `applicationName` in KCL-Version 3.x oder höher aufgerufen wird. Einzelheiten finden Sie unter Kompilierungsfehler mit dem LeaseManagementConfig Konstruktor. Wir empfehlen die Verwendung`ConfigsBuilder`, um KCL-Konfigurationen festzulegen. `ConfigsBuilder`bietet eine flexiblere und wartungsfreundlichere Methode zur Konfiguration Ihrer KCL-Anwendung.

Schritt 4: Folgen Sie den bewährten Methoden für die Implementierung der Methode shutdownRequested ()

Mit KCL 3.x wird eine Funktion namens Graceful Lease Handoff eingeführt, um die erneute Verarbeitung von Daten zu minimieren, wenn ein Mietvertrag im Rahmen der Neuzuweisung eines Leasingvertrags an einen anderen Mitarbeiter übergeben wird. Dies wird erreicht, indem die zuletzt verarbeitete Sequenznummer in der Leasetabelle vor der Leasing-Übergabe überprüft wird. Um sicherzustellen, dass die ordnungsgemäße Lease-Übergabe ordnungsgemäß funktioniert, müssen Sie sicherstellen, dass Sie das `checkpointer` Objekt innerhalb der `shutdownRequested` Methode in Ihrer Klasse aufrufen. `RecordProcessor` Wenn Sie das `checkpointer` Objekt nicht innerhalb der `shutdownRequested` Methode aufrufen, können Sie es wie im folgenden Beispiel dargestellt implementieren.

> **Important**
>
> - Das folgende Implementierungsbeispiel ist eine Mindestanforderung für die reibungslose Leasing-Übergabe. Sie können es bei Bedarf um zusätzliche Logik im Zusammenhang mit

dem Checkpointing erweitern. Wenn Sie eine asynchrone Verarbeitung durchführen, stellen Sie sicher, dass alle an den Downstream übermittelten Datensätze verarbeitet wurden, bevor Sie Checkpointing aufrufen.

- Eine reibungslose Leasing-Übergabe verringert zwar die Wahrscheinlichkeit einer erneuten Verarbeitung von Daten bei Leasing-Übertragungen erheblich, schließt diese Möglichkeit jedoch nicht vollständig aus. Um Datenintegrität und Konsistenz zu wahren, sollten Sie Ihre nachgelagerten Verbraucheranwendungen so gestalten, dass sie idempotent sind. Das bedeutet, dass sie in der Lage sein sollten, die Verarbeitung potenzieller doppelter Datensätze ohne nachteilige Auswirkungen auf das Gesamtsystem zu bewältigen.

```
/**
 * Invoked when either Scheduler has been requested to gracefully shutdown
 * or lease ownership is being transferred gracefully so the current owner
 * gets one last chance to checkpoint.
 *
 * Checkpoints and logs the data a final time.
 *
 * @param shutdownRequestedInput Provides access to a checkpointer, allowing a record
 processor to checkpoint
 *                               before the shutdown is completed.
 */
public void shutdownRequested(ShutdownRequestedInput shutdownRequestedInput) {
    try {
       // Ensure that all delivered records are processed
       // and has been successfully flushed to the downstream before calling
       // checkpoint
       // If you are performing any asynchronous processing or flushing to
       // downstream, you must wait for its completion before invoking
       // the below checkpoint method.
        log.info("Scheduler is shutting down, checkpointing.");
        shutdownRequestedInput.checkpointer().checkpoint();
    } catch (ShutdownException | InvalidStateException e) {
        log.error("Exception while checkpointing at requested shutdown. Giving up.",
 e);
    }
}
```

Schritt 5: Überprüfen Sie die KCL 3.x-Voraussetzungen für die Erfassung von Arbeitskräftekennzahlen

KCL 3.x sammelt Metriken zur CPU-Auslastung, wie z. B. die CPU-Auslastung von Mitarbeitern, um die Auslastung gleichmäßig auf die Mitarbeiter verteilen zu können. Anwendungsarbeiter für Privatanwender können auf Amazon EC2, Amazon ECS, Amazon EKS oder AWS Fargate ausgeführt werden. KCL 3.x kann Metriken zur CPU-Auslastung von Workern nur erfassen, wenn die folgenden Voraussetzungen erfüllt sind:

Amazon Elastic Compute Cloud(Amazon EC2)

- Ihr Betriebssystem muss Linux OS sein.
- Sie müssen es IMDSv2in Ihrer EC2-Instance aktivieren.

Amazon Elastic Container Service (Amazon ECS) auf Amazon EC2

- Ihr Betriebssystem muss Linux OS sein.
- Sie müssen den ECS-Endpunkt für Task-Metadaten, Version 4, aktivieren.
- Ihre Amazon ECS-Container-Agent-Version muss 1.39.0 oder höher sein.

Amazon ECS auf AWS Fargate

- Sie müssen den Fargate-Task-Metadaten-Endpunkt Version 4 aktivieren. Wenn Sie die Fargate-Plattformversion 1.4.0 oder höher verwenden, ist dies standardmäßig aktiviert.
- Fargate-Plattformversion 1.4.0 oder höher.

Amazon Elastic Kubernetes Service (Amazon EKS) auf Amazon EC2

- Ihr Betriebssystem muss Linux OS sein.

Amazon EKS auf AWS Fargate

- Fargate-Plattform 1.3.0 oder höher.

> **Important**
>
> Wenn KCL 3.x keine CPU-Auslastungsmetriken von Workern sammeln kann, weil die Voraussetzungen nicht erfüllt sind, verteilt es die Last auf das Durchsatzniveau pro Lease. Dieser Ausweichmechanismus stellt sicher, dass alle Mitarbeiter einen ähnlichen Gesamtdurchsatz aus den Leasingverträgen erhalten, die jedem Mitarbeiter zugewiesen wurden. Weitere Informationen finden Sie unter Wie KCL Mitarbeitern Leasingverträge zuweist und die Auslastung verteilt.

Schritt 6: Aktualisieren Sie die IAM-Berechtigungen für KCL 3.x

Sie müssen der IAM-Rolle oder -Richtlinie, die mit Ihrer KCL 3.x-Verbraucheranwendung verknüpft ist, die folgenden Berechtigungen hinzufügen. Dies beinhaltet die Aktualisierung der vorhandenen IAM-Richtlinie, die von der KCL-Anwendung verwendet wird. Weitere Informationen finden Sie unter Für KCL-Anwendungen für Privatanwender sind IAM-Berechtigungen erforderlich.

> **Important**
>
> In Ihren vorhandenen KCL-Anwendungen wurden die folgenden IAM-Aktionen und Ressourcen möglicherweise nicht zur IAM-Richtlinie hinzugefügt, da sie in KCL 2.x nicht benötigt wurden. Stellen Sie sicher, dass Sie sie hinzugefügt haben, bevor Sie Ihre KCL 3.x-Anwendung ausführen:
>
> - Aktionen: `UpdateTable`
>   - Ressourcen (ARNs): `arn:aws:dynamodb:region:account:table/KCLApplicationName`
> - Aktionen: `Query`
>   - Ressourcen (ARNs): `arn:aws:dynamodb:region:account:table/KCLApplicationName/index/*`
> - Aktionen:`CreateTable`, `DescribeTableScan`,`GetItem`,`PutItem`,`UpdateItem`, `DeleteItem`
>   - Ressourcen (ARNs):`arn:aws:dynamodb:region:account:table/KCLApplicationName-WorkerMetricStats`, `arn:aws:dynamodb:region:account:table/KCLApplicationName-CoordinatorState`

Ersetzen Sie „Region“, „Konto“ und „KCLApplicationName“ in der ARNs durch Ihre eigene AWS-Region AWS-Konto Nummer bzw. den Namen der KCL-Anwendung. Wenn Sie Konfigurationen verwenden, um die Namen der von KCL erstellten Metadatentabellen anzupassen, verwenden Sie diese angegebenen Tabellennamen anstelle des KCL-Anwendungsnamens.

Schritt 7: Stellen Sie Ihren Mitarbeitern KCL 3.x-Code bereit

Nachdem Sie die für die Migration erforderliche Konfiguration festgelegt und alle vorherigen Migrationschecklisten abgeschlossen haben, können Sie Ihren Code erstellen und für Ihre Mitarbeiter bereitstellen.

Note

Wenn Sie einen Kompilierungsfehler mit dem `LeaseManagementConfig` Konstruktor sehen, finden Sie Informationen zur Problembehandlung unter Kompilierungsfehler mit dem LeaseManagementConfig Konstruktor.

Schritt 8: Schließen Sie die Migration ab

Während der Bereitstellung von KCL 3.x-Code verwendet KCL weiterhin den Lease-Zuweisungsalgorithmus aus KCL 2.x. Wenn Sie erfolgreich KCL 3.x-Code für alle Ihre Mitarbeiter bereitgestellt haben, erkennt KCL dies automatisch und wechselt auf den neuen Algorithmus für die Leasingzuweisung, der auf der Ressourcennutzung der Mitarbeiter basiert. Weitere Informationen zum neuen Algorithmus für die Leasingzuweisung finden Sie unter. Wie KCL Mitarbeitern Leasingverträge zuweist und die Auslastung verteilt

Während der Bereitstellung können Sie den Migrationsprozess anhand der folgenden Metriken überwachen, die an gesendet CloudWatch werden. Sie können die Metriken während des `Migration` Vorgangs überwachen. Bei allen Metriken handelt es sich um per-KCL-application Metriken, die auf `SUMMARY` Metrik-Ebene festgelegt sind. Wenn die `Sum` Statistik der `CurrentState:3xWorker` Metrik mit der Gesamtzahl der Mitarbeiter in Ihrer KCL-Anwendung übereinstimmt, bedeutet dies, dass die Migration zu KCL 3.x erfolgreich abgeschlossen wurde.

> **Important**
>
> Es dauert mindestens 10 Minuten, bis KCL auf den neuen Algorithmus für die Zuweisung von Mietverträgen umgestellt hat, nachdem alle Mitarbeiter bereit sind, ihn auszuführen.

CloudWatch Metriken für den KCL-Migrationsprozess

Kennzahlen	Description
`CurrentState:3xWorker`	Die Anzahl der KCL-Worker, die erfolgreich auf KCL 3.x migriert wurden und den neuen Leasing-Zuweisungsalgorithmus ausführen . Wenn die `Sum` Anzahl dieser Metrik mit der Gesamtzahl Ihrer Mitarbeiter übereinstimmt, bedeutet dies, dass die Migration zu KCL 3.x erfolgreich abgeschlossen wurde. • Metrikstufe: Summary • Einheiten: Anzahl • Statistik: Die nützlichste Statistik ist `Sum`
`CurrentState:2xCompatibleWorker`	Die Anzahl der KCL-Worker, die während des Migrationsprozesses im KCL 2.x-kompa tiblen Modus ausgeführt werden. Ein Wert ungleich Null für diese Metrik bedeutet, dass die Migration noch im Gange ist. • Metrikstufe: Summary • Einheiten: Anzahl • Statistik: Die nützlichste Statistik ist `Sum`
`Fault`	Die Anzahl der Ausnahmen, die während des Migrationsprozesses aufgetreten sind. Bei den meisten dieser Ausnahmen handelt es sich um vorübergehende Fehler, und KCL 3.x versucht automatisch erneut, die Migration abzuschli eßen. Wenn Sie einen dauerhaften `Fault`

Kennzahlen	Description
	Metrikwert beobachten, überprüfen Sie Ihre Protokolle aus dem Migrationszeitraum zur weiteren Fehlerbehebung. Wenn das Problem weiterhin besteht, wenden Sie sich an Support. • Metrikstufe: Summary • Einheiten: Anzahl • Statistik: Die nützlichste Statistik ist `Sum`
`GsiStatusReady`	Der Status der Erstellung des globalen Sekundärindex (GSI) in der Leasingtabelle. Diese Metrik gibt an, ob der GSI in der Leasetabelle erstellt wurde. Dies ist eine Voraussetzung für die Ausführung von KCL 3.x. Der Wert ist 0 oder 1, wobei 1 für eine erfolgrei che Erstellung steht. Während eines Rollback-Status wird diese Metrik nicht ausgegebe n. Nachdem Sie erneut einen Rollforward durchgeführt haben, können Sie die Überwachu ng dieser Metrik fortsetzen. • Metrikstufe: Summary • Einheiten: Anzahl • Statistik: Die nützlichste Statistik ist `Sum`

Kennzahlen	Description
`workerMetricsReady`	Status der Arbeitnehmer misst die Emissione n aller Arbeitnehmer. Die Metriken geben an, ob alle Mitarbeiter Kennzahlen wie die CPU-Auslastung ausgeben. Der Wert ist 0 oder 1, wobei 1 bedeutet, dass alle Mitarbeiter erfolgrei ch Kennzahlen ausgeben und für den neuen Algorithmus zur Leasingzuweisung bereit sind. Während eines Rollback-Status wird diese Metrik nicht ausgegeben. Nachdem Sie erneut einen Rollforward durchgeführt haben, können Sie die Überwachung dieser Metrik fortsetzen. • Metrikstufe: Summary • Einheiten: Anzahl • Statistik: Die nützlichste Statistik ist `Sum`

KCL bietet während der Migration die Möglichkeit, auf den 2.x-kompatiblen Modus zurückzugreifen. Nach erfolgreicher Migration auf KCL 3.x empfehlen wir Ihnen, die `CoordinatorConfig.clientVersionConfig` Einstellung „`CLIENT_VERSION_CONFIG_COMPATIBLE_WITH_2X`Wenn ein Rollback nicht mehr erforderlich ist“ zu entfernen. Durch das Entfernen dieser Konfiguration wird die Ausgabe migrationsbezogener Metriken aus der KCL-Anwendung gestoppt.

> **Note**
>
> Wir empfehlen Ihnen, die Leistung und Stabilität Ihrer Anwendung für einen bestimmten Zeitraum während der Migration und nach Abschluss der Migration zu überwachen. Wenn Sie Probleme feststellen, können Sie Workers mithilfe des KCL-Migrationstools auf die Verwendung von KCL 2.x-kompatiblen Funktionen zurücksetzen.

Rollback zu einer früheren KCL-Version

In diesem Thema werden die Schritte erläutert, mit denen Sie Ihren Kunden auf die vorherige Version zurücksetzen können. Wenn Sie ein Rollback durchführen müssen, erfolgt dies in zwei Schritten:

1. Führen Sie das KCL Migration Tool aus.
2. Stellen Sie den Code der vorherigen KCL-Version erneut bereit (optional).

Schritt 1: Ausführen des KCL Migration Tools

Wenn Sie zur vorherigen KCL-Version zurückkehren möchten, müssen Sie das KCL Migration Tool ausführen. Das KCL-Migrationstool erfüllt zwei wichtige Aufgaben:

- Es entfernt die Metadatentabelle für Worker-Metriken und den globalen sekundären Index aus der Leasetabelle in DynamoDB. Diese beiden Artefakte wurden von KCL 3.x erstellt, werden aber nicht benötigt, wenn Sie zur vorherigen Version zurückkehren.
- Dadurch werden alle Worker in einem mit KCL 2.x kompatiblen Modus ausgeführt und beginnen, den Load-Balancing-Algorithmus zu verwenden, der in früheren KCL-Versionen verwendet wurde. Eventuelle Probleme mit dem neuen Load-Balancing-Algorithmus in KCL 3.x werden dadurch sofort behoben.

> **Important**
>
> Die Koordinatorstatustabelle in DynamoDB muss vorhanden sein und darf während des Migrations-, Rollback- und Rollforward-Prozesses nicht gelöscht werden.

> **Note**
>
> Es ist wichtig, dass alle Worker in Ihrer Verbraucheranwendung zu einem bestimmten Zeitpunkt denselben Load-Balancing-Algorithmus verwenden. Das KCL-Migrationstool stellt sicher, dass alle Worker in Ihrer KCL 3.x-Consumer-Anwendung in den KCL 2.x-kompatiblen Modus wechseln, sodass alle Mitarbeiter während der Rückzahlung auf Ihre vorherige KCL-Version denselben Load-Balancing-Algorithmus verwenden.

Sie können das KCL-Migrationstool im Skriptverzeichnis des KCL-Repositorys herunterladen. GitHub Das Skript kann von jedem Ihrer Worker oder jedem Host ausgeführt werden, der über die erforderlichen Berechtigungen verfügt, um in die Tabelle mit den Koordinatorstatus zu schreiben, die Tabelle mit den Worker-Metriken zu löschen und die Leasing-Tabelle zu aktualisieren. Die erforderlichen IAM-Berechtigungen Für KCL-Anwendungen für Privatanwender sind IAM-

Berechtigungen erforderlich zur Ausführung des Skripts finden Sie unter. Sie dürfen das Skript nur einmal pro KCL-Anwendung ausführen. Sie können das KCL-Migrationstool mit dem folgenden Befehl ausführen:

```
python3 ./KclMigrationTool.py --region <region> --mode rollback [--
application_name <applicationName>] [--lease_table_name <leaseTableName>] [--
coordinator_state_table_name <coordinatorStateTableName>] [--worker_metrics_table_name
 <workerMetricsTableName>]
```

Parameter

- --region: Ersetze es `<region>` durch dein. AWS-Region
- --application_name: Dieser Parameter ist erforderlich, wenn Sie Standardnamen für Ihre DynamoDB-Metadatentabellen (Leasetabelle, Koordinatorstatustabelle und Worker-Metriktabelle) verwenden. Wenn Sie benutzerdefinierte Namen für diese Tabellen festgelegt haben, können Sie diesen Parameter weglassen. Ersetzen Sie es durch Ihren `<applicationName>` tatsächlichen KCL-Anwendungsnamen. Das Tool verwendet diesen Namen, um die Standardtabellennamen abzuleiten, wenn keine benutzerdefinierten Namen angegeben werden.
- --lease_table_name (optional): Dieser Parameter wird benötigt, wenn Sie in Ihrer KCL-Konfiguration einen benutzerdefinierten Namen für die Leasetabelle festgelegt haben. Wenn Sie den Standardtabellennamen verwenden, können Sie diesen Parameter weglassen. `leaseTableName`Ersetzen Sie ihn durch den benutzerdefinierten Tabellennamen, den Sie für Ihre Leasingtabelle angegeben haben.
- --coordinator_state_table_name (optional): Dieser Parameter wird benötigt, wenn Sie in Ihrer KCL-Konfiguration einen benutzerdefinierten Namen für die Koordinatorstatentabelle festgelegt haben. Wenn Sie den Standardtabellennamen verwenden, können Sie diesen Parameter weglassen. Ersetzen Sie ihn durch den benutzerdefinierten `<coordinatorStateTableName>` Tabellennamen, den Sie für Ihre Koordinatorstatustabelle angegeben haben.
- --worker_metrics_table_name (optional): Dieser Parameter wird benötigt, wenn Sie in Ihrer KCL-Konfiguration einen benutzerdefinierten Namen für die Tabelle mit den Worker-Metriken festgelegt haben. Wenn Sie den Standardtabellennamen verwenden, können Sie diesen Parameter weglassen. Ersetzen Sie ihn durch den benutzerdefinierten `<workerMetricsTableName>` Tabellennamen, den Sie für Ihre Worker-Metriktabelle angegeben haben.

Schritt 2: Stellen Sie den Code erneut mit der vorherigen KCL-Version bereit (optional)

Nachdem das KCL Migration Tool den Rollback ausgeführt hat, wird eine der folgenden Meldungen angezeigt:

- Meldung 1: „Rollback abgeschlossen. Ihre KCL-Anwendung lief im KCL 2.x-kompatiblen Modus. Wenn Sie keine Abmilderung der Regression sehen, kehren Sie bitte zu Ihren vorherigen Anwendungsbinärdateien zurück, indem Sie den Code mit Ihrer vorherigen KCL-Version bereitstellen.“
  - Erforderliche Maßnahme: Das bedeutet, dass Ihre Mitarbeiter im KCL 2.x-kompatiblen Modus gearbeitet haben. Wenn das Problem weiterhin besteht, stellen Sie den Code mit der vorherigen KCL-Version erneut für Ihre Mitarbeiter bereit.
- Meldung 2: „Rollback abgeschlossen. In Ihrer KCL-Anwendung wurde der KCL 3.x-Funktionsmodus ausgeführt. Ein Rollback zu den vorherigen Anwendungsbinärdateien ist nicht erforderlich, es sei denn, Sie sehen innerhalb von 5 Minuten keine Lösung für das Problem. Wenn Sie immer noch ein Problem haben, führen Sie bitte ein Rollback zu Ihren vorherigen Anwendungsbinärdateien durch, indem Sie den Code mit Ihrer vorherigen KCL-Version bereitstellen.“
  - Erforderliche Maßnahme: Das bedeutet, dass Ihre Worker im KCL 3.x-Modus liefen und das KCL-Migrationstool alle Worker in den KCL 2.x-kompatiblen Modus umgestellt hat. Wenn das Problem behoben ist, müssen Sie den Code nicht erneut mit der vorherigen KCL-Version bereitstellen. Wenn das Problem weiterhin besteht, stellen Sie den Code mit der vorherigen KCL-Version erneut für Ihre Mitarbeiter bereit.

Rollforward zu KCL 3.x nach einem Rollback

In diesem Thema werden die Schritte erläutert, mit denen Sie Ihren Kunden nach einem Rollback wieder auf KCL 3.x umstellen können. Wenn Sie ein Rollforward durchführen müssen, müssen Sie einen zweistufigen Prozess durchlaufen:

1. Führen Sie das KCL Migration Tool aus.
2. Stellen Sie den Code mit KCL 3.x bereit.

Schritt 1: Ausführen des KCL Migration Tools

Führen Sie das KCL Migration Tool aus. KCL-Migrationstool mit dem folgenden Befehl für den Rollforward auf KCL 3.x:

```
python3 ./KclMigrationTool.py --region <region> --mode rollforward [--application_name
 <applicationName>] [--coordinator_state_table_name <coordinatorStateTableName>]
```

Parameter

- --region: Ersetze es durch dein. `<region>` AWS-Region
- --application_name: Dieser Parameter ist erforderlich, wenn Sie Standardnamen für Ihre Koordinatorstatentabelle verwenden. Wenn Sie benutzerdefinierte Namen für diese Tabelle festgelegt haben, können Sie diesen Parameter weglassen. Ersetzen Sie ihn `<applicationName>` durch Ihren tatsächlichen KCL-Anwendungsnamen. Das Tool verwendet diesen Namen, um die Standardtabellennamen abzuleiten, wenn keine benutzerdefinierten Namen angegeben werden.
- --coordinator_state_table_name (optional): Dieser Parameter wird benötigt, wenn Sie in Ihrer KCL-Konfiguration einen benutzerdefinierten Namen für die Koordinatorstatentabelle festgelegt haben. Wenn Sie den Standardtabellennamen verwenden, können Sie diesen Parameter weglassen. Ersetzen Sie ihn durch den benutzerdefinierten `<coordinatorStateTableName>` Tabellennamen, den Sie für Ihre Koordinatorstatustabelle angegeben haben.

Nachdem Sie das Migrationstool im Roll-Forward-Modus ausgeführt haben, erstellt KCL die folgenden DynamoDB-Ressourcen, die für KCL 3.x erforderlich sind:

- einen globalen sekundären Index für die Leasetabelle
- eine Tabelle mit Worker-Metriken

Schritt 2: Bereitstellen des Codes mit KCL 3.x

Nachdem Sie den Rollforward mit dem KCL Migration Tool ausgeführt haben, stellen Sie Ihren Code mit KCL 3.x für Ihre Worker bereit. Folgen Sie Schritt 8: Schließen Sie die Migration ab den Anweisungen, um Ihre Migration abzuschließen.

Bewährte Methoden für die Leasetabelle mit dem Modus „Bereitgestellte Kapazität“

Wenn die Leasetabelle Ihrer KCL-Anwendung in den Modus für bereitgestellte Kapazität umgestellt wurde, erstellt KCL 3.x einen globalen sekundären Index in der Leasetabelle mit dem bereitgestellten Abrechnungsmodus und denselben Lesekapazitätseinheiten (RCU) und Schreibkapazitätseinheiten (WCU) wie in der Basis-Leasetabelle. Wenn der globale sekundäre Index erstellt wird, empfehlen wir, die tatsächliche Nutzung des globalen sekundären Index in der DynamoDB-Konsole zu überwachen und die Kapazitätseinheiten bei Bedarf anzupassen. Eine ausführlichere Anleitung zum Umschalten des Kapazitätsmodus von DynamoDB-Metadatentabellen, die von KCL erstellt wurden, finden Sie unter. DynamoDB-Kapazitätsmodus für von KCL erstellte Metadatentabellen

> **Note**
>
> Standardmäßig erstellt KCL im On-Demand-Kapazitätsmodus Metadatentabellen wie die Leasetabelle, die Tabelle mit den Worker-Metriken und den Koordinatorstatus-Tabellen sowie den globalen Sekundärindex für die Leasing-Tabelle. Wir empfehlen, den On-Demand-Kapazitätsmodus zu verwenden, um die Kapazität automatisch an Ihre Nutzungsänderungen anzupassen.

Migrieren von KCL 1.x zu KCL 3.x

In diesem Thema werden die Anweisungen zur Migration Ihres Verbrauchers von KCL 1.x auf KCL 3.x erläutert. KCL 1.x verwendet im Vergleich zu KCL 2.x und KCL 3.x andere Klassen und Schnittstellen. Sie müssen zuerst die Klassen Record Processor, Record Processor Factory und Worker auf das KCL 2.x/3.x-kompatible Format migrieren und dann die Migrationsschritte für die Migration von KCL 2.x zu KCL 3.x befolgen. Sie können direkt von KCL 1.x auf KCL 3.x aktualisieren.

- Schritt 1: Migrieren Sie den Aufnahmeprozessor

  Folgen Sie dem Abschnitt Den Datensatzprozessor migrieren auf der Seite Verbraucher von KCL 1.x zu KCL 2.x migrieren.

- Schritt 2: Migrieren Sie den Record Processor ab Werk

  Folgen Sie auf der Seite Verbraucher von KCL 1.x auf KCL 2.x migrieren im Abschnitt Werkseinstellungen für den Record Processor migrieren.

- Schritt 3: Migrieren Sie den Worker

Folgen Sie dem Abschnitt Den Worker migrieren auf der Seite Verbraucher von KCL 1.x zu KCL 2.x migrieren.

- Schritt 4: Migrieren Sie die KCL 1.x-Konfiguration

  Folgen Sie dem Abschnitt Amazon Kinesis Kinesis-Client konfigurieren auf der Seite Verbraucher von KCL 1.x zu KCL 2.x migrieren.

- Schritt 5: Überprüfen Sie die Entfernung der Leerlaufzeiten und die Entfernung der Client-Konfiguration

  Folgen Sie den Abschnitten Entfernung von Leerlaufzeiten und Entfernen der Client-Konfiguration auf der Seite Verbraucher von KCL 1.x zu KCL 2.x migrieren.

- Schritt 6: Folgen Sie den step-by-step Anweisungen im Migrationshandbuch von KCL 2.x zu KCL 3.x

  Folgen Sie den Anweisungen auf der Migrieren Sie von KCL 2.x zu KCL 3.x Seite, um die Migration abzuschließen. Wenn Sie nach einem Rollback zur vorherigen KCL-Version zurückkehren oder ein Rollforward zu KCL 3.x durchführen müssen, finden Sie weitere Informationen unter und. Rollback zu einer früheren KCL-Version Rollforward zu KCL 3.x nach einem Rollback

> **Important**
>
> Verwenden Sie die AWS SDK für Java Versionen 2.27.19 bis 2.27.23 nicht mit KCL 3.x. Diese Versionen enthalten ein Problem, das einen Ausnahmefehler im Zusammenhang mit der DynamoDB-Nutzung von KCL verursacht. Wir empfehlen, die AWS SDK für Java Version 2.28.0 oder höher zu verwenden, um dieses Problem zu vermeiden.

Dokumentation zu früheren KCL-Versionen

Die folgenden Themen wurden archiviert. Die aktuelle Dokumentation zur Kinesis Client Library finden Sie unterVerwenden Sie die Kinesis-Clientbibliothek.

> **Important**
>
> Die Versionen 1.x und 2.x der Amazon Kinesis Client Library (KCL) sind veraltet. KCL 1.x wird end-of-support am 30. Januar 2026 verfügbar sein. Wir empfehlen dringend, dass Sie Ihre KCL-Anwendungen, die Version 1.x verwenden, vor dem 30. Januar 2026 auf die

neueste KCL-Version migrieren. Die neueste KCL-Version finden Sie auf der Seite Amazon Kinesis Client Library unter. GitHub Informationen zu den neuesten KCL-Versionen finden Sie unter. Verwenden Sie die Kinesis-Clientbibliothek Informationen zur Migration von KCL 1.x zu KCL 3.x finden Sie unter. Migrieren von KCL 1.x zu KCL 3.x

Dokumentation wurde eingestellt

- Informationen zu KCL 1.x und 2.x
- Entwickeln Sie benutzerdefinierte Verbraucher mit gemeinsamem Durchsatz
- Migrieren Sie Verbraucher von KCL 1.x zu KCL 2.x

Informationen zu KCL 1.x und 2.x

Important

Die Versionen 1.x und 2.x der Amazon Kinesis Client Library (KCL) sind veraltet. KCL 1.x wird am 30. Januar 2026 end-of-support verfügbar sein. Wir empfehlen dringend, dass Sie Ihre KCL-Anwendungen, die Version 1.x verwenden, vor dem 30. Januar 2026 auf die neueste KCL-Version migrieren. Die neueste KCL-Version finden Sie auf der Seite Amazon Kinesis Client Library unter. GitHub Informationen zu den neuesten KCL-Versionen finden Sie unter. Verwenden Sie die Kinesis-Clientbibliothek Informationen zur Migration von KCL 1.x zu KCL 3.x finden Sie unter. Migrieren von KCL 1.x zu KCL 3.x

Eine der Methoden zur Entwicklung benutzerdefinierter Verbraucheranwendungen, die Daten aus KDS-Datenströmen verarbeiten können, ist die Verwendung der Kinesis Client Library (KCL).

Themen

- Über KCL (frühere Versionen)
- Frühere Versionen von KCL
- KCL-Konzepte (frühere Versionen)
- Verwenden Sie eine Leasetabelle, um nachzuverfolgen, welche Shards von der KCL-Consumer-Anwendung verarbeitet wurden
- Verarbeiten Sie mehrere Datenströme mit derselben KCL 2.x für Java-Consumer-Anwendung
- Verwenden Sie die KCL mit der Schema Registry AWS Glue

> **Note**
>
> Sowohl für KCL 1.x als auch für KCL 2.x wird empfohlen, je nach Nutzungsszenario ein Upgrade auf die neueste KCL-1.x-Version oder KCL-2.x-Version durchzuführen. Sowohl KCL 1.x als auch KCL 2.x werden regelmäßig mit neueren Versionen aktualisiert, die die neuesten Abhängigkeits- und Sicherheitspatches, Bugfixes und abwärtskompatible neue Features enthalten. Weitere Informationen finden Sie unter /releases. https://github.com/awslabs/amazon-kinesis-client

Über KCL (frühere Versionen)

KCL unterstützt Sie bei der Nutzung und Verarbeitung von Daten aus einem Kinesis-Datenstrom, indem sie sich um viele der komplexen Aufgaben kümmert, die mit verteilter Datenverarbeitung verbunden sind. Dazu gehören der Lastenausgleich über mehrere Anwendungs-Instances hinweg, die Reaktion auf Ausfälle von Verbraucheranwendungs-Instances, die Überprüfung verarbeiteter Datensätze und die Reaktion auf Resharding. Die KCL kümmert sich um all diese Unteraufgaben, sodass Sie sich darauf konzentrieren können, Ihre benutzerdefinierte Logik für die Verarbeitung von Datensätzen zu schreiben.

Die KCL unterscheidet sich von den Kinesis Data Streams APIs , die in der verfügbar sind. AWS SDKs Die Kinesis Data Streams APIs helfen Ihnen bei der Verwaltung vieler Aspekte von Kinesis Data Streams, darunter das Erstellen von Streams, das Resharding sowie das Einfügen und Abrufen von Datensätzen. Die KCL bietet eine Abstraktionsebene für all diese Unteraufgaben, sodass Sie sich auf die benutzerdefinierte Datenverarbeitungslogik Ihrer Verbraucheranwendung konzentrieren können. Informationen zur Kinesis-Data-Streams-API finden Sie in der Amazon-Kinesis-API-Referenz.

> **Important**
>
> Die KCL ist eine Java-Bibliothek. Die Support für andere Sprachen als Java wird über eine mehrsprachige Schnittstelle namens bereitgestellt. MultiLangDaemon Dieser Daemon basiert auf Java und wird im Hintergrund ausgeführt, wenn Sie eine andere KCL-Sprache als Java verwenden. Wenn Sie beispielsweise die KCL für Python installieren und Ihre Consumer-Anwendung vollständig in Python schreiben, müssen Sie trotzdem Java auf Ihrem System installiert haben, da MultiLangDaemon Darüber hinaus MultiLangDaemon verfügt es über einige Standardeinstellungen, die Sie möglicherweise an Ihren Anwendungsfall anpassen müssen, z. B. an die AWS Region, mit der eine Verbindung hergestellt wird. Weitere

Informationen dazu finden Sie unter MultiLangDaemon KCL-Projekt. MultiLangDaemon GitHub

Die KCL dient als Vermittler zwischen Ihrer Datensatzverarbeitungslogik und Kinesis Data Streams.

Frühere Versionen von KCL

Derzeit können Sie eine der folgenden unterstützten Versionen von KCL verwenden, um Ihre benutzerdefinierten Anwendungen für Privatanwender zu erstellen:

- KCL 1.x

  Weitere Informationen finden Sie unter Entwickeln Sie KCL 1.x-Verbraucher.
- KCL 2.x

  Weitere Informationen finden Sie unter Entwickeln Sie KCL 2.x-Verbraucher.

Sie können entweder KCL 1.x oder KCL 2.x verwenden, um Verbraucheranwendungen zu erstellen, die einen gemeinsamen Durchsatz verwenden. Weitere Informationen finden Sie unter Entwickeln Sie mithilfe von KCL benutzerdefinierte Verbraucher mit gemeinsamem Durchsatz.

Um Verbraucheranwendungen zu erstellen, die einen dedizierten Durchsatz verwenden (Enhanced Fan-Out-Consumer), können Sie nur KCL 2.x verwenden. Weitere Informationen finden Sie unter Entwickeln Sie verbesserte Fan-Out-Verbraucher mit dediziertem Durchsatz.

Informationen zu den Unterschieden zwischen KCL 1.x und KCL 2.x sowie Anweisungen zur Migration von KCL 1.x zu KCL 2.x finden Sie unter Migrieren Sie Verbraucher von KCL 1.x zu KCL 2.x.

KCL-Konzepte (frühere Versionen)

- KCL-Konsumentenanwendung – eine Anwendung, die unter Verwendung von KCL kundenspezifisch entwickelt wurde und zum Lesen und Verarbeiten von Datensätzen aus Datenströmen bestimmt ist.
- Konsumentenanwendungs-Instance – KCL-Konsumentenanwendungen werden in der Regel verteilt, wobei eine oder mehrere Anwendungs-Instances gleichzeitig ausgeführt werden, um Fehler zu koordinieren und einen dynamischen Lastenausgleich bei der Verarbeitung von Datensätzen vorzunehmen.

- Worker – eine übergeordnete Klasse, die eine KCL-Konsumentenanwendungs-Instance verwendet, um mit der Datenverarbeitung zu beginnen.

  > **Important**
  >
  > Jede KCL-Konsumentenanwendungs-Instance hat einen Worker.

  Der Worker initialisiert und überwacht verschiedene Aufgaben, darunter das Synchronisieren von Shard- und Leasing-Informationen, das Verfolgen von Shard-Zuweisungen und das Verarbeiten von Daten aus den Shards. Ein Worker stellt KCL die Konfigurationsinformationen für die Consumer-Anwendung zur Verfügung, z. B. den Namen des Datenstroms, dessen Datensätze diese KCL-Consumer-Anwendung verarbeiten wird, und die AWS Anmeldeinformationen, die für den Zugriff auf diesen Datenstrom erforderlich sind. Der Worker startet außerdem die spezifische Instance der KCL-Konsumentenanwendung, um Datensätze aus dem Datenstrom an die Datensatzprozessoren zu übermitteln.

  > **Important**
  >
  > In KCL 1.x heißt diese Klasse Worker. Weitere Informationen (dies sind die Java-KCL-Repositorys) finden Sie unter/.java. https://github.com/awslabs/ amazon-kinesis-client blob/v1.x/src/main/java/com/amazonaws/services/kinesis/clientlibrary/lib/worker/Worker In KCL 2.x heißt diese Klasse Scheduler. Der Zweck von Scheduler in KCL 2.x ist identisch mit dem Zweck von Worker in KCL 1.x. Weitere Informationen zur Scheduler-Klasse in KCL 2.x finden Sie unter/.java. https://github.com/awslabs/ amazon-kinesis-client blob/master/amazon-kinesis-client/src/main/java/software/amazon/kinesis/coordinator/Scheduler

- Lease – Daten, die die Bindung zwischen einem Worker und einem Shard definieren. Verteilte KCL-Konsumentenanwendungen nutzen Leases, um die Verarbeitung von Datensätzen auf eine ganze Flotte von Workern zu verteilen. Zu einem bestimmten Zeitpunkt ist jeder Shard von Datensätzen durch einen Lease, der durch die leaseKey-Variable identifiziert wird, an einen bestimmten Worker gebunden.

  Standardmäßig kann ein Mitarbeiter einen oder mehrere Leasingverträge (abhängig vom Wert der Variablen maxLeasesForWorker) gleichzeitig abschließen.

> **Important**
>
> Jeder Worker versucht, alle verfügbaren Leases für alle verfügbaren Shards in einem Datenstrom zu halten. Aber es kann nur jeweils ein Worker jeden Lease zu einem bestimmten Zeitpunkt erfolgreich halten.

Wenn Sie beispielsweise eine Konsumentenanwendungs-Instance A mit Worker A haben, die einen Datenstrom mit 4 Shards verarbeitet, kann Worker A Leases für die Shards 1, 2, 3 und 4 gleichzeitig halten. Wenn Sie jedoch zwei Konsumentenanwendungs-Instances haben: A und B mit Worker A und Worker B und diese Instances einen Datenstrom mit 4 Shards verarbeiten, können Worker A und Worker B nicht gleichzeitig den Lease für Shard 1 halten. Ein Worker hält den Lease für einen bestimmten Shard, bis er bereit ist, die Verarbeitung der Datensätze dieses Shards zu beenden, oder bis er ausfällt. Wenn ein Worker den Lease beendet, nimmt ein anderer Worker den Lease auf und hält ihn.

Weitere Informationen (dies sind die Java-KCL-Repositorys) finden Sie unter https://github.com/awslabs/amazon-kinesis-client/blob/v1.x/src/main/java/com/amazonaws/services/kinesis/leases/impl/Lease.java für KCL 1.x und https://github.com/awslabs/amazon-kinesis-client/.java für KCL 2.x. blob/master/amazon-kinesis-client/src/main/java/software/amazon/kinesis/leases/Lease

- Leasetabelle – Eine einzigartige Amazon-DynamoDB-Tabelle, die verwendet wird, um die Shards in einem KDS-Datenstrom zu verfolgen, die von den Workern der KCL-Konsumentenanwendung geleast und verarbeitet werden. Die Leasetabelle muss (innerhalb eines Workers und für alle Worker) mit den neuesten Shard-Informationen aus dem Datenstrom synchronisiert bleiben, während die KCL-Konsumentenanwendung ausgeführt wird. Weitere Informationen finden Sie unter Verwenden Sie eine Leasetabelle, um nachzuverfolgen, welche Shards von der KCL-Consumer-Anwendung verarbeitet wurden.
- Datensatzprozessor – Die Logik, die definiert, wie Ihre KCL-Konsumentenanwendung die Daten verarbeitet, die sie aus den Datenströmen erhält. Zur Laufzeit instanziiert eine KCL-Konsumentenanwendungs-Instance einen Worker, und dieser Worker instanziiert einen Datensatzprozessor für jeden Shard, für den er einen Lease hat.

Verwenden Sie eine Leasetabelle, um nachzuverfolgen, welche Shards von der KCL-Consumer-Anwendung verarbeitet wurden

Themen

- Was ist eine Leasingtabelle
- Durchsatz
- So wird eine Leasing-Tabelle mit den Shards in einem Kinesis-Datenstrom synchronisiert

Was ist eine Leasingtabelle

Für jede Anwendung mit Amazon Kinesis Data Streams verwendet KCL eine einzigartige Leasetabelle (gespeichert in einer Amazon-DynamoDB-Tabelle), die verwendet wird, um die Shards in einem KDS-Datenstrom zu verfolgen, die von den Workern der KCL-Konsumentenanwendung geleast und verarbeitet werden.

> **Important**
>
> KCL verwendet den Namen der Konsumentenanwendung, um den Namen der Leasetabelle zu erstellen, die diese Konsumentenanwendung verwendet. Daher muss der Name jeder Konsumentenanwendung eindeutig sein.

Sie können die Leasetabelle mit der Amazon-DynamoDB-Konsole anzeigen, während die Konsumentenanwendung ausgeführt wird.

Wenn die Leasetabelle für Ihre KCL-Konsumentenanwendung beim Start der Anwendung nicht vorhanden ist, erstellt einer der Worker die Leasetabelle für diese Anwendung.

> **Important**
>
> Ihr Konto wird neben den Kosten für Kinesis Data Streams mit den Kosten belastet, die für die DynamoDB-Tabelle anfallen.

Jede Zeile in der Leasetabelle stellt einen Shard dar, der von den Workern Ihrer Konsumentenanwendung verarbeitet wird. Wenn Ihre KCL-Konsumentenanwendung nur einen Datenstrom verarbeitet, dann ist `leaseKey`, der Hash-Schlüssel für die Leasetabelle, die Shard-ID. Wenn Sie Verarbeiten Sie mehrere Datenströme mit derselben KCL 2.x für Java-Consumer-Anwendung sind, dann sieht die Struktur des leaseKey wie folgt aus: `account-id:StreamName:streamCreationTimestamp:ShardId`. Beispiel, `111111111:multiStreamTest-1:12345:shardId-000000000336`.

Neben der Shard-ID enthält jede Zeile noch folgende Daten:

- checkpoint: Die letzte Prüfpunkt-Sequenznummer des Shards. Dieser Wert ist für alle Shards im Datenstrom eindeutig.
- checkpointSubSequenceNummer: Wenn Sie die Aggregationsfunktion der Kinesis Producer Library verwenden, ist dies eine Erweiterung des Checkpoints, mit der einzelne Benutzerdatensätze innerhalb des Kinesis-Datensatzes verfolgt werden.
- leaseCounter: Wird für ein Lease Versioning verwendet, damit Auftragnehmer erkennen, wenn ihr Lease von einem anderen Auftragnehmer übernommen wurde.
- leaseKey: Eine eindeutige Kennung für einen Lease. Jeder Lease gilt für einen bestimmten Shard im Datenstrom und wird immer nur von einem Worker gehalten.
- leaseOwner: Der Auftragnehmer, dem der Lease gehört.
- ownerSwitchesSinceCheckpoint: Wie oft hat dieser Leasingvertrag seit der letzten Erstellung eines Checkpoints die Mitarbeiter gewechselt?
- parentShardId: Wird verwendet, um sicherzustellen, dass der übergeordnete Shard vollständig verarbeitet ist, bevor die Verarbeitung der untergeordneten Shards beginnt. So wird sichergestellt, dass Datensätze in der gleichen Reihenfolge verarbeitet werden, in der sie in den Stream eingespeist wurden.
- hashrange: Wird von `PeriodicShardSyncManager` verwendet, um regelmäßige Synchronisierungen durchzuführen, um fehlende Shards in der Leasetabelle zu finden und bei Bedarf Leases für sie zu erstellen.

  > **Note**
  >
  > Diese Daten sind in der Leasetabelle für jeden Shard enthalten, der mit KCL 1.14 und KCL 2.3 beginnt. Weitere Hinweise zu `PeriodicShardSyncManager` und die periodische Synchronisierung zwischen Leases und Shards finden Sie unter So wird eine Leasing-Tabelle mit den Shards in einem Kinesis-Datenstrom synchronisiert.

- childshards: Wird von `LeaseCleanupManager` verwendet, um den Verarbeitungsstatus des untergeordneten Shards zu überprüfen und zu entscheiden, ob der übergeordnete Shard aus der Leasetabelle gelöscht werden kann.

> **Note**
>
> Diese Daten sind in der Leasetabelle für jeden Shard enthalten, der mit KCL 1.14 und KCL 2.3 beginnt.

- shardID: Die ID des Shards.

> **Note**
>
> Diese Daten sind nur dann in der Leasetabelle enthalten, wenn Sie Verarbeiten Sie mehrere Datenströme mit derselben KCL 2.x für Java-Consumer-Anwendung sind. Dies wird nur in KCL 2.x für Java unterstützt, beginnend mit KCL 2.3 für Java und höher.

- Streamname Die Kennung des Datenstroms im folgenden Format: `account-id:StreamName:streamCreationTimestamp`.

> **Note**
>
> Diese Daten sind nur dann in der Leasetabelle enthalten, wenn Sie Verarbeiten Sie mehrere Datenströme mit derselben KCL 2.x für Java-Consumer-Anwendung sind. Dies wird nur in KCL 2.x für Java unterstützt, beginnend mit KCL 2.3 für Java und höher.

Durchsatz

Wenn Ihre Amazon Kinesis Data Streams-Anwendung Ausnahmen für den bereitgestellten Durchsatz erhält, sollten Sie den bereitgestellten Durchsatz für die DynamoDB-Tabelle erhöhen. Die KCL erstellt die Tabelle mit einem bereitgestellten Durchsatz von 10 Lese- und 10 Schreibvorgängen pro Sekunde, dies reicht aber möglichweise für Ihre Anwendung nicht aus. Beispiel: Wenn Ihre Amazon Kinesis Data Streams-Anwendung häufig Prüfpunkte setzt oder einen Stream verarbeitet, der aus vielen Shards besteht, müssen Sie den Durchsatz möglicherweise erhöhen.

Weitere Informationen zum bereitgestellten Durchsatz in DynamoDB finden Sie unter Lese-/Schreibkapazitätsmodus und Arbeiten mit Tabellen und Daten im Amazon-DynamoDB-Entwicklerhandbuch.

So wird eine Leasing-Tabelle mit den Shards in einem Kinesis-Datenstrom synchronisiert

Worker in KCL-Konsumentenanwendungen verwenden Leases, um Shards aus einem bestimmten Datenstrom zu verarbeiten. Die Informationen darüber, welcher Worker zu einem bestimmten Zeitpunkt welchen Shard least, werden in einer Leasetabelle gespeichert. Die Leasetabelle muss mit den neuesten Shard-Informationen aus dem Datenstrom synchronisiert bleiben, während die KCL-Konsumentenanwendung ausgeführt wird. KCL synchronisiert die Leasetabelle mit den Shard-Informationen, die während des Bootstrapings der Konsumentenanwendung (entweder wenn die Konsumentenanwendung initialisiert oder neu gestartet wird) vom Dienst Kinesis Data Streams erfasst werden und auch immer dann, wenn ein Shard, der gerade verarbeitet wird, ein Ende erreicht (Resharding), erreicht. Mit anderen Worten, die Worker oder eine KCL-Konsumentenanwendung werden mit dem Datenstrom synchronisiert, den sie beim ersten Bootstrap der Konsumentenanwendung verarbeiten, und immer dann wenn die Konsumentenanwendung auf ein Reshard-Ereignis des Datenstroms trifft.

Themen

- Synchronisation in KCL 1.0–1.13 und KCL 2.0–2.2
- Synchronisation in KCL 2.x, beginnend mit KCL 2.3 und höher
- Synchronisation in KCL 1.x, beginnend mit KCL 1.14 und höher

Synchronisation in KCL 1.0–1.13 und KCL 2.0–2.2

In KCL 1.0-1.13 und KCL 2.0-2.2 synchronisiert KCL beim Bootstraping der Verbraucheranwendung und auch bei jedem Reshard-Ereignis des Datenstroms die Leasetabelle mit den Shard-Informationen, die vom Kinesis Data Streams Streams-Dienst abgerufen wurden, indem die oder die Erkennung aufgerufen wird. `ListShards` `DescribeStream` APIs In allen oben aufgeführten KCL-Versionen führt jeder Worker einer KCL-Consumer-Anwendung die folgenden Schritte durch, um den lease/shard Synchronisationsprozess während des Bootstrappings der Consumer-Anwendung und bei jedem Stream-Reshard-Ereignis durchzuführen:

- Ruft alle Shards für Daten des Streams ab, der gerade verarbeitet wird
- Ruft alle Shard-Leases aus der Leasetabelle ab
- Filtert jeden offenen Shard heraus, für den es in der Leasetabelle keinen Lease gibt
- Iteriert über alle gefundenen offenen Shards und für jeden offenen Shard ohne offenes übergeordnetes Element:

- Durchläuft den Hierarchiebaum über den Pfad der Vorgänger, um festzustellen, ob der Shard ein Nachfolger ist. Ein Shard wird als untergeordnetes Element betrachtet, wenn gerade ein Vorgänger-Shard verarbeitet wird (der Leaseeintrag für den Vorgänger-Shard ist in der Leasetabelle vorhanden) oder wenn ein Vorgänger-Shard verarbeitet werden soll (wenn die Anfangsposition beispielsweise `TRIM_HORIZON` oder `AT_TIMESTAMP` ist)
- Handelt es sich bei dem offenen Shard im Kontext um einen Nachfolger-Shard, überprüft KCL den Shard anhand seiner Ausgangsposition und erstellt, falls erforderlich, Leaseverträge für die übergeordneten Shard

Synchronisation in KCL 2.x, beginnend mit KCL 2.3 und höher

Beginnend mit den neuesten unterstützten Versionen von KCL 2.x (KCL 2.3) und höher unterstützt die Bibliothek nun die folgenden Änderungen am Synchronisationsprozess. Diese lease/shard Synchronisierungsänderungen reduzieren die Anzahl der API-Aufrufe, die von KCL-Consumer-Anwendungen an den Kinesis Data Streams Streams-Service getätigt werden, erheblich und optimieren das Leasing-Management in Ihrer KCL-Consumer-Anwendung.

- Wenn beim Bootstraping der Anwendung die Leasetabelle leer ist, verwendet KCL die Filteroption der `ListShard`-API (den optionalen Anforderungsparameter von `ShardFilter`), um Leases nur für einen Snapshot von Shards abzurufen und zu erstellen, die zu dem durch den `ShardFilter`-Parameter angegebenen Zeitpunkt geöffnet waren. Mit dem `ShardFilter`-Parameter können Sie die Antwort der `ListShards`-API herausfiltern. Die einzige erforderliche Eigenschaft des `ShardFilter`-Parameters ist `Type`. KCL verwendet die `Type`-Filtereigenschaft und die folgenden gültigen Werte, um offene Shards, für die möglicherweise neue Leases erforderlich sind, zu identifizieren und eine Momentaufnahme zurückzugeben:
  - `AT_TRIM_HORIZON` – Die Antwort umfasst alle Shards, die am `TRIM_HORIZON` geöffnet waren.
  - `AT_LATEST` – Die Antwort enthält nur die aktuell geöffneten Shards des Datenstroms.
  - `AT_TIMESTAMP` – Die Antwort umfasst alle Shards, deren Startzeitstempel kleiner oder gleich dem angegebenen Zeitstempel sind und deren Endzeitstempel größer oder gleich dem angegebenen Zeitstempel ist oder solche, die noch offen sind.

  `ShardFilter` wird verwendet, wenn Leases für eine leere Leasetabelle erstellt werden, um Leases für einen Snapshot von Shards zu initialisieren, die unter `RetrievalConfig#initialPositionInStreamExtended` angegeben sind.

  Mehr über `ShardFilter` erfahren Sie unter https://docs.aws.amazon.com/kinesis/latest/APIReference/API_ShardFilter.html.

- Anstatt dass alle Worker die lease/shard Synchronisation durchführen, um die Leasetabelle mit den neuesten Shards im Datenstrom auf dem neuesten Stand zu halten, führt ein einziger gewählter Worker-Leiter die Leasing-/Shard-Synchronisierung durch.
- KCL 2.3 verwendet den `ChildShards` Rückgabeparameter von `GetRecords` und, `SubscribeToShard` APIs um die lease/shard Synchronisation durchzuführen, die bei geschlossenen Shards stattfindet, sodass ein KCL-Worker nur Leases `SHARD_END` für die untergeordneten Shards des Shards erstellen kann, dessen Verarbeitung abgeschlossen ist. Bei Anwendungen, die von allen Benutzern gemeinsam genutzt werden, verwendet diese Optimierung der lease/shard Synchronisation den API-Parameter. `ChildShards` `GetRecords` Für Verbraucheranwendungen mit dediziertem Durchsatz (erweiterter Fan-Out) verwendet diese Optimierung der lease/shard Synchronisation den `ChildShards` `SubscribeToShard` API-Parameter. Weitere Informationen finden Sie unter GetRecords, SubscribeToShards und ChildShard.
- Mit den oben genannten Änderungen ändert sich das Verhalten von KCL von dem Modell, dass alle Worker über alle vorhandenen Shards lernen, hin zu dem Modell, dass Worker nur noch über die untergeordneten Shards der Shards lernen, die jeder Worker besitzt. Daher führt KCL jetzt zusätzlich zu der Synchronisation, die beim Bootstraping von Verbraucheranwendungen und bei Reshard-Ereignissen stattfindet, auch zusätzliche regelmäßige shard/lease Scans durch, um potenzielle Lücken in der Leasetabelle zu identifizieren (mit anderen Worten, um mehr über alle neuen Shards zu erfahren), um sicherzustellen, dass der gesamte Hash-Bereich des Datenstroms verarbeitet wird, und um bei Bedarf Leases für sie zu erstellen. `PeriodicShardSyncManager`ist die Komponente, die für die regelmäßige Ausführung von Scans verantwortlich ist. lease/shard

  Weitere Informationen zu `PeriodicShardSyncManager` KCL 2.3 finden Sie unter https://github.com/awslabs/amazon-kinesis-client/blob/master/amazon-kinesis-client/src/main/java/software/amazon/kinesis/leases/LeaseManagementConfig.java #L201 -L213.

  In KCL 2.3 stehen neue Konfigurationsoptionen, um `PeriodicShardSyncManager` in `LeaseManagementConfig` zu konfigurieren:

Name	Standardwert	Description
leasesRecoveryAuditorExecutionFrequencyMillis	120 000 (2 Minuten)	Häufigkeit (in Millionen), mit der der Auditor in der Leasetabelle

Name	Standardwert	Description
		nach teilweise n Leases sucht. Wenn der Auditor eine Lücke in den Leases für einen Stream feststellt, würde er die Shard-Synchronisie rung auf der Grundlage von `leasesRec overyAudi torIncons istencyCo nfidenceT hreshold` auslösen.

Name	Standardwert	Description
leasesRecoveryAuditorInconsistencyConfidenceThreshold	3	Konfidenzschwellenwert für die regelmäßige Auditortätigkeit, um festzustellen, ob die Leases für einen Datenstrom in der Leasetabelle inkonsistent sind. Wenn der Auditor für diesen Datenstrom mehrmals hintereinander dieselben Inkonsistenzen feststellt, würde er eine Shard-Synchronisierung auslösen.

Außerdem werden jetzt neue CloudWatch Messwerte ausgegeben, um den Zustand von zu überwachen. `PeriodicShardSyncManager` Weitere Informationen finden Sie unter PeriodicShardSyncManager.

- Einschließlich einer Optimierung auf `HierarchicalShardSyncer`, um nur Leases für eine Shard-Ebene zu erstellen.

Synchronisation in KCL 1.x, beginnend mit KCL 1.14 und höher

Beginnend mit den neuesten unterstützten Versionen von KCL 1.x (KCL 1.14) und höher unterstützt die Bibliothek nun die folgenden Änderungen am Synchronisationsprozess. Diese lease/shard

Synchronisierungsänderungen reduzieren die Anzahl der API-Aufrufe, die von KCL-Consumer-Anwendungen an den Kinesis Data Streams Streams-Service getätigt werden, erheblich und optimieren das Leasing-Management in Ihrer KCL-Consumer-Anwendung.

- Wenn beim Bootstraping der Anwendung die Leasetabelle leer ist, verwendet KCL die Filteroption der `ListShard`-API (den optionalen Anforderungsparameter von `ShardFilter`), um Leases nur für einen Snapshot von Shards abzurufen und zu erstellen, die zu dem durch den `ShardFilter`-Parameter angegebenen Zeitpunkt geöffnet waren. Mit dem `ShardFilter`-Parameter können Sie die Antwort der `ListShards`-API herausfiltern. Die einzige erforderliche Eigenschaft des `ShardFilter`-Parameters ist `Type`. KCL verwendet die `Type`-Filtereigenschaft und die folgenden gültigen Werte, um offene Shards, für die möglicherweise neue Leases erforderlich sind, zu identifizieren und eine Momentaufnahme zurückzugeben:
  - `AT_TRIM_HORIZON` – Die Antwort umfasst alle Shards, die am `TRIM_HORIZON` geöffnet waren.
  - `AT_LATEST` – Die Antwort enthält nur die aktuell geöffneten Shards des Datenstroms.
  - `AT_TIMESTAMP` – Die Antwort umfasst alle Shards, deren Startzeitstempel kleiner oder gleich dem angegebenen Zeitstempel sind und deren Endzeitstempel größer oder gleich dem angegebenen Zeitstempel ist oder solche, die noch offen sind.

  `ShardFilter` wird verwendet, wenn Leases für eine leere Leasetabelle erstellt werden, um Leases für einen Snapshot von Shards zu initialisieren, die unter `KinesisClientLibConfiguration#initialPositionInStreamExtended` angegeben sind.

  Mehr über `ShardFilter` erfahren Sie unter https://docs.aws.amazon.com/kinesis/latest/APIReference/API_ShardFilter.html.
- Anstatt dass alle Worker die lease/shard Synchronisation durchführen, um die Leasetabelle mit den neuesten Shards im Datenstrom auf dem neuesten Stand zu halten, führt ein einziger gewählter Worker-Leiter die Leasing-/Shard-Synchronisierung durch.
- KCL 1.14 verwendet den `ChildShards` Rückgabeparameter von `GetRecords` und, `SubscribeToShard` APIs um die lease/shard Synchronisation durchzuführen, die bei geschlossenen Shards stattfindet, sodass ein KCL-Worker nur Leases `SHARD_END` für die untergeordneten Shards des Shards erstellen kann, dessen Verarbeitung abgeschlossen ist. Weitere Informationen erhalten Sie unter GetRecords und ChildShard.
- Mit den oben genannten Änderungen ändert sich das Verhalten von KCL von dem Modell, dass alle Worker über alle vorhandenen Shards lernen, hin zu dem Modell, dass Worker nur noch über die untergeordneten Shards der Shards lernen, die jeder Worker besitzt. Daher führt KCL jetzt

zusätzlich zu der Synchronisation, die beim Bootstraping von Verbraucheranwendungen und bei Reshard-Ereignissen stattfindet, auch zusätzliche regelmäßige shard/lease Scans durch, um potenzielle Lücken in der Leasetabelle zu identifizieren (mit anderen Worten, um mehr über alle neuen Shards zu erfahren), um sicherzustellen, dass der gesamte Hash-Bereich des Datenstroms verarbeitet wird, und um bei Bedarf Leases für sie zu erstellen. `PeriodicShardSyncManager`ist die Komponente, die für die regelmäßige Ausführung von Scans verantwortlich ist. lease/shard

Wenn `KinesisClientLibConfiguration#shardSyncStrategyType` auf `ShardSyncStrategyType.SHARD_END` gesetzt ist, wird `PeriodicShardSync` `leasesRecoveryAuditorInconsistencyConfidenceThreshold` verwendet, um den Schwellenwert für die Anzahl der aufeinanderfolgenden Scans mit Lücken in der Leasetabelle zu bestimmen, nach dem eine Shard-Synchronisierung erzwungen werden soll. Wenn `KinesisClientLibConfiguration#shardSyncStrategyType` auf `ShardSyncStrategyType.PERIODIC` gesetzt ist, wird `leasesRecoveryAuditorInconsistencyConfidenceThreshold` ignoriert.

Weitere Informationen zu `PeriodicShardSyncManager` KCL 1.14 finden Sie unter https://github.com/awslabs/amazon-kinesis-client/blob/v1.x/src/main/java/com/amazonaws/services/kinesis/clientlibrary/lib/worker/KinesisClientLibConfiguration.java #L987 -L999.

In KCL 1.14 ist eine neue Konfigurationsoption verfügbar, um `PeriodicShardSyncManager` in `LeaseManagementConfig` zu konfigurieren:

Name	Standardwert	Description
leasesRecoveryAuditorInconsistencyConfidenceThreshold	3	Konfidenzschwellenwert für die regelmäßige Auditortätigkeit, um festzustellen, ob die Leases für einen Datenstrom in der Leasetabelle inkonsistent sind. Wenn der Auditor für diesen Datenstrom mehrmals hintereinander dieselben Inkonsistenzen feststellt, würde er eine Shard-Synchronisierung auslösen.

Außerdem werden jetzt neue CloudWatch Messwerte ausgegeben, um den Zustand von zu überwachen. `PeriodicShardSyncManager` Weitere Informationen finden Sie unter PeriodicShardSyncManager.

- KCL 1.14 unterstützt jetzt auch die verzögerte Lease-Bereinigung. Leases werden asynchron von `LeaseCleanupManager` bei Erreichen von `SHARD_END` gelöscht, wenn ein Shard entweder die Aufbewahrungsfrist des Datenstroms überschritten hat oder wenn er aufgrund eines Resharding-Vorgangs geschlossen wurde.

Es stehen neue Konfigurationsoptionen zur Verfügung, um `LeaseCleanupManager` zu konfigurieren.

Name	Standardwert	Description
leaseCleanupIntervalMillis	1 Minute	Intervall, in dem der Lease-Cleanup-Thread ausgeführt werden soll.
completedLeaseCleanupIntervalMillis	5 Minuten	Intervall, in dem überprüft werden soll, ob ein Lease abgeschlossen ist oder nicht.
garbageLeaseCleanupIntervalMillis	30 Minuten	Intervall, in dem geprüft werden soll, ob ein Lease Datenmüll ist (d. h. über die Aufbewahrungsfrist des Datenstroms hinaus abgeschnitten wurde) oder nicht.

- Einschließlich einer Optimierung auf `KinesisShardSyncer`, um nur Leases für eine Shard-Ebene zu erstellen.

Verarbeiten Sie mehrere Datenströme mit derselben KCL 2.x für Java-Consumer-Anwendung

In diesem Abschnitt werden die folgenden Änderungen in KCL 2.x für Java beschrieben, mit denen Sie KCL-Verbraucheranwendungen erstellen können, die mehr als einen Datenstrom gleichzeitig verarbeiten können.

> **Important**
>
> Die Multistream-Verarbeitung wird nur in KCL 2.x für Java unterstützt, beginnend mit KCL 2.3 für Java und höher.
> Die Multistream-Verarbeitung wird NICHT für andere Sprachen unterstützt, in denen KCL 2.x implementiert werden kann.
> Die Multistream-Verarbeitung wird in keiner Version von KCL 1.x unterstützt.

- MultistreamTracker Schnittstelle

  Um eine Verbraucheranwendung zu erstellen, die mehrere Streams gleichzeitig verarbeiten kann, müssen Sie eine neue Schnittstelle namens implementieren MultistreamTracker. Diese Schnittstelle enthält die `streamConfigList`-Methode, die die Liste der Datenströme und ihrer Konfigurationen zurückgibt, die von der KCL-Konsumentenanwendung verarbeitet werden sollen. Beachten Sie, dass die Datenströme, die verarbeitet werden, während der Laufzeit der Konsumentenanwendung geändert werden können. `streamConfigList` wird regelmäßig von der KCL aufgerufen, um mehr über die Änderungen der zu verarbeitenden Datenströme zu erfahren.

  Die `streamConfigList` Methode füllt die StreamConfigListe auf.

```
package software.amazon.kinesis.common;

import lombok.Data;
import lombok.experimental.Accessors;

@Data
@Accessors(fluent = true)
public class StreamConfig {
    private final StreamIdentifier streamIdentifier;
    private final InitialPositionInStreamExtended initialPositionInStreamExtended;
    private String consumerArn;
}
```

Beachten Sie, dass es sich bei den Feldern `StreamIdentifier` und `InitialPositionInStreamExtended` um Pflichtfelder handelt, während `consumerArn` optional ist. Sie müssen den `consumerArn` nur angeben, wenn Sie KCL 2.x verwenden, um eine erweiterte Fan-Out-Konsumentenanwendung zu implementieren.

Weitere Informationen dazu finden Sie `StreamIdentifier` unter https://github.com/awslabs/amazon-kinesis-client/blob/v2.5.8/amazon-kinesis-client/src/main/java/software/amazon/kinesis/common/StreamIdentifier.java #L129. Um eine zu erstellen`StreamIdentifier`, empfehlen wir, dass Sie eine Multistream-Instanz aus der `streamArn` und der erstellen`streamCreationEpoch`, die in Version 2.5.0 und höher verfügbar ist. Erstellen Sie in KCL v2.3 und v2.4, die dies nicht unterstützen`streamArm`, eine Multistream-Instanz mithilfe des folgenden Formats. `account-id:StreamName:streamCreationTimestamp` Dieses Format ist veraltet und wird ab der nächsten Hauptversion nicht mehr unterstützt.

`MultistreamTracker` beinhaltet auch eine Strategie zum Löschen von Leases alter Streams in der Leasetabelle (`formerStreamsLeasesDeletionStrategy`). Beachten Sie, dass die Strategie während der Laufzeit der Konsumentenanwendung NICHT geändert werden kann. Weitere Informationen finden Sie unter https://github.com/awslabs/ amazon-kinesis-client / blob/0c5042dadf794fe988438436252a5a8fe70b6b0 .java b/amazon-kinesis-client/src/main/java/software/amazon/kinesis/processor/FormerStreamsLeasesDeletionStrategy

- ConfigsBuilderist eine anwendungsweite Klasse, mit der Sie alle KCL 2.x-Konfigurationseinstellungen angeben können, die beim Erstellen Ihrer KCL-Consumer-Anwendung verwendet werden sollen. `ConfigsBuilder`Die Klasse unterstützt jetzt die Schnittstelle. `MultistreamTracker` Sie können ConfigsBuilder entweder mit dem Namen des einen Datenstroms initialisieren, aus dem Datensätze abgerufen werden sollen:

```
/**
    * Constructor to initialize ConfigsBuilder with StreamName
    * @param streamName
    * @param applicationName
    * @param kinesisClient
    * @param dynamoDBClient
    * @param cloudWatchClient
    * @param workerIdentifier
    * @param shardRecordProcessorFactory
    */
```

```
   public ConfigsBuilder(@NonNull String streamName, @NonNull String
 applicationName,
           @NonNull KinesisAsyncClient kinesisClient, @NonNull DynamoDbAsyncClient
 dynamoDBClient,
           @NonNull CloudWatchAsyncClient cloudWatchClient, @NonNull String
 workerIdentifier,
           @NonNull ShardRecordProcessorFactory shardRecordProcessorFactory) {
       this.appStreamTracker = Either.right(streamName);
       this.applicationName = applicationName;
       this.kinesisClient = kinesisClient;
       this.dynamoDBClient = dynamoDBClient;
       this.cloudWatchClient = cloudWatchClient;
       this.workerIdentifier = workerIdentifier;
       this.shardRecordProcessorFactory = shardRecordProcessorFactory;
   }
```

Oder Sie können ConfigsBuilder mit initialisieren, `MultiStreamTracker` wenn Sie eine KCL-Consumer-Anwendung implementieren möchten, die mehrere Streams gleichzeitig verarbeitet.

```
* Constructor to initialize ConfigsBuilder with MultiStreamTracker
     * @param multiStreamTracker
     * @param applicationName
     * @param kinesisClient
     * @param dynamoDBClient
     * @param cloudWatchClient
     * @param workerIdentifier
     * @param shardRecordProcessorFactory
     */
   public ConfigsBuilder(@NonNull MultiStreamTracker multiStreamTracker, @NonNull
 String applicationName,
           @NonNull KinesisAsyncClient kinesisClient, @NonNull DynamoDbAsyncClient
 dynamoDBClient,
           @NonNull CloudWatchAsyncClient cloudWatchClient, @NonNull String
 workerIdentifier,
           @NonNull ShardRecordProcessorFactory shardRecordProcessorFactory) {
       this.appStreamTracker = Either.left(multiStreamTracker);
       this.applicationName = applicationName;
       this.kinesisClient = kinesisClient;
       this.dynamoDBClient = dynamoDBClient;
       this.cloudWatchClient = cloudWatchClient;
       this.workerIdentifier = workerIdentifier;
       this.shardRecordProcessorFactory = shardRecordProcessorFactory;
```

```
    }
```

- Da die Multistream-Unterstützung für Ihre KCL-Konsumentenanwendung implementiert ist, enthält jede Zeile der Leasetabelle der Anwendung jetzt die Shard-ID und den Stream-Namen der mehreren Datenströme, die diese Anwendung verarbeitet.
- Wenn die Multistream-Unterstützung für Ihre KCL-Konsumentenanwendung implementiert ist, hat der leaseKey die folgende Struktur: `account-id:StreamName:streamCreationTimestamp:ShardId`. Beispiel, `111111111:multiStreamTest-1:12345:shardId-000000000336`.

> **Important**
>
> Wenn Ihre vorhandene KCL-Konsumentenanwendung so konfiguriert ist, dass sie nur einen Datenstrom verarbeitet, ist der leaseKey (der Hash-Schlüssel für die Leasetabelle) die Shard-ID. Wenn Sie diese vorhandene KCL-Konsumentenanwendung für die Verarbeitung mehrerer Datenströme neu konfigurieren, wird Ihre Leasetabelle beschädigt, da bei Multistream-Unterstützung die leaseKey-Struktur wie folgt aussehen muss: `account-id:StreamName:StreamCreationTimestamp:ShardId`.

Verwenden Sie die KCL mit der Schema Registry AWS Glue

Sie können Ihre Kinesis-Datenströme in die AWS Glue Schema Registry integrieren. Mit der AWS Glue Schema Registry können Sie Schemas zentral erkennen, steuern und weiterentwickeln und gleichzeitig sicherstellen, dass die erstellten Daten kontinuierlich anhand eines registrierten Schemas validiert werden. Ein Schema definiert die Struktur und das Format eines Datensatzes. Ein Schema ist eine versionierte Spezifikation für zuverlässige Datenveröffentlichung, -nutzung oder -speicherung. Mit der AWS Glue Schema Registry können Sie die end-to-end Datenqualität und Datenverwaltung in Ihren Streaming-Anwendungen verbessern. Weitere Informationen finden Sie unter AWS Glue Schema Registry. Eine Möglichkeit, diese Integration einzurichten, ist die KCL in Java.

> **Important**
>
> Derzeit wird die Integration von Kinesis Data Streams und AWS Glue Schema Registry nur für Kinesis-Datenstreams unterstützt, die in Java implementierte KCL 2.3-Consumer verwenden. Mehrsprachige Unterstützung wird nicht bereitgestellt. KCL-1.0-Konsumenten werden nicht unterstützt. KCL-2.x-Konsumenten vor KCL 2.3 werden nicht unterstützt.

Detaillierte Anweisungen zur Einrichtung der Integration von Kinesis Data Streams mit Schema Registry mithilfe der KCL finden Sie im Abschnitt „Interaktion mit Daten mithilfe der KPL/KCL Bibliotheken“ unter Anwendungsfall: Integration von Amazon Kinesis Data Streams mit der AWS Glue Schema Registry.

Entwickeln Sie benutzerdefinierte Verbraucher mit gemeinsamem Durchsatz

> **Important**
>
> Die Versionen 1.x und 2.x der Amazon Kinesis Client Library (KCL) sind veraltet. KCL 1.x wird end-of-support am 30. Januar 2026 verfügbar sein. Wir empfehlen dringend, dass Sie Ihre KCL-Anwendungen, die Version 1.x verwenden, vor dem 30. Januar 2026 auf die neueste KCL-Version migrieren. Die neueste KCL-Version finden Sie auf der Seite Amazon Kinesis Client Library unter. GitHub Informationen zu den neuesten KCL-Versionen finden Sie unter. Verwenden Sie die Kinesis-Clientbibliothek Informationen zur Migration von KCL 1.x zu KCL 3.x finden Sie unter. Migrieren von KCL 1.x zu KCL 3.x

Wenn Sie keinen spezifischen Durchsatz beim Empfangen von Daten von Kinesis Data Streams und keine Verbreitungswerte für Leseoperationen von unter 200 ms benötigen, können Sie Konsumentenanwendungen wie in den folgenden Themen beschrieben erstellen. Sie können die Kinesis Client Library (KCL) oder die verwenden. AWS SDK für Java

Themen

- Entwickeln Sie mithilfe von KCL benutzerdefinierte Verbraucher mit gemeinsamem Durchsatz

Für Informationen zum Erstellen von Verbrauchern, die Datensätze aus Kinesis-Datenströmen mit dediziertem Durchsatz empfangen können, siehe Entwickeln Sie verbesserte Fan-Out-Verbraucher mit dediziertem Durchsatz.

Entwickeln Sie mithilfe von KCL benutzerdefinierte Verbraucher mit gemeinsamem Durchsatz

> **Important**
>
> Die Versionen 1.x und 2.x der Amazon Kinesis Client Library (KCL) sind veraltet. KCL 1.x wird end-of-support am 30. Januar 2026 verfügbar sein. Wir empfehlen dringend, dass Sie Ihre KCL-Anwendungen, die Version 1.x verwenden, vor dem 30. Januar 2026 auf die neueste KCL-Version migrieren. Die neueste KCL-Version finden Sie auf der Seite Amazon

Kinesis Client Library unter. GitHub Informationen zu den neuesten KCL-Versionen finden Sie unter. Verwenden Sie die Kinesis-Clientbibliothek Informationen zur Migration von KCL 1.x zu KCL 3.x finden Sie unter. Migrieren von KCL 1.x zu KCL 3.x

Eine der Methoden zur Entwicklung einer benutzerdefinierten Verbraucheranwendung mit gemeinsamem Durchsatz ist die Verwendung der Kinesis Client Library (KCL).

Wählen Sie eines der folgenden Themen für die KCL-Version, die Sie verwenden.

Themen

- Entwickeln Sie KCL 1.x-Verbraucher
- Entwickeln Sie KCL 2.x-Verbraucher

Entwickeln Sie KCL 1.x-Verbraucher

Important

Die Versionen 1.x und 2.x der Amazon Kinesis Client Library (KCL) sind veraltet. KCL 1.x wird am 30. Januar 2026 end-of-support verfügbar sein. Wir empfehlen dringend, dass Sie Ihre KCL-Anwendungen, die Version 1.x verwenden, vor dem 30. Januar 2026 auf die neueste KCL-Version migrieren. Die neueste KCL-Version finden Sie auf der Seite Amazon Kinesis Client Library unter. GitHub Informationen zu den neuesten KCL-Versionen finden Sie unter. Verwenden Sie die Kinesis-Clientbibliothek Informationen zur Migration von KCL 1.x zu KCL 3.x finden Sie unter. Migrieren von KCL 1.x zu KCL 3.x

Sie können eine Verbraucheranwendung für Amazon Kinesis Data Streams entwickeln, indem Sie die Kinesis Client Library (KCL) verwenden.

Weitere Informationen zu KCL finden Sie unter Über KCL (frühere Versionen).

Wählen Sie je nach der Option, die Sie verwenden möchten, aus den folgenden Themen.

Inhalt

- Entwickeln Sie einen Kinesis Client Library-Consumer in Java
- Entwickeln Sie einen Kinesis Client Library-Consumer in Node.js
- Entwickeln Sie einen Kinesis Client Library-Consumer in .NET

- Entwickeln Sie einen Kinesis Client Library-Consumer in Python
- Entwickeln Sie einen Kinesis Client Library-Consumer in Ruby

Entwickeln Sie einen Kinesis Client Library-Consumer in Java

> **Important**
>
> Die Versionen 1.x und 2.x der Amazon Kinesis Client Library (KCL) sind veraltet. KCL 1.x wird end-of-support am 30. Januar 2026 verfügbar sein. Wir empfehlen dringend, dass Sie Ihre KCL-Anwendungen, die Version 1.x verwenden, vor dem 30. Januar 2026 auf die neueste KCL-Version migrieren. Die neueste KCL-Version finden Sie auf der Seite Amazon Kinesis Client Library unter. GitHub Informationen zu den neuesten KCL-Versionen finden Sie unter. Verwenden Sie die Kinesis-Clientbibliothek Informationen zur Migration von KCL 1.x zu KCL 3.x finden Sie unter. Migrieren von KCL 1.x zu KCL 3.x

Sie können die Kinesis Client Library (KCL) verwenden, um Anwendungen zu erstellen, die Daten aus Ihren Kinesis-Datenströmen verarbeiten. Die Kinesis Client Library ist in mehreren Sprachen verfügbar. In diesem Thema wird Java behandelt. Die Javadoc-Referenz finden Sie im Javadoc-Thema für Class.AWS AmazonKinesisClient

Gehen Sie zur Kinesis Client Library (Java) GitHub, um die Java-KCL von herunterzuladen. Um die Java KCL auf Apache Maven zu finden, navigieren Sie zur Seite für die KCL-Suchergebnisse. Um Beispielcode für eine Java-KCL-Verbraucheranwendung herunterzuladen GitHub, gehen Sie auf die Projektseite KCL for Java-Beispielprojekt unter. GitHub

Die Beispielanwendung verwendet Apache Commons Logging. Sie können die Konfiguration der Protokollierung in der statischen Methode `configure` ändern, die in der Datei `AmazonKinesisApplicationSample.java` definiert ist. Weitere Informationen zur Verwendung von Apache Commons Logging mit Log4j und AWS Java-Anwendungen finden Sie unter Logging with Log4j im Developer Guide.AWS SDK für Java

Sie müssen die folgenden Aufgaben durchführen, wenn Sie eine KCL-Konsumentenanwendung in Java implementieren:

Aufgaben

- Implementieren Sie die Methoden des Prozessors IRecord
- Implementieren Sie eine Klassenfabrik für die IRecord Prozessorschnittstelle

- Erstellen Sie einen Worker
- Ändern Sie die Konfigurationseigenschaften
- Migrieren Sie zu Version 2 der Record Processor-Schnittstelle

Implementieren Sie die Methoden des Prozessors IRecord

Die KCL unterstützt zurzeit zwei Versionen der `IRecordProcessor`-Schnittstelle: die ursprüngliche Schnittstelle, die mit der ersten Version der KCL verfügbar war, und Version 2, die ab KCL Version 1.5.0 verfügbar ist. Beide Schnittstellen werden vollständig unterstützt. Ihre Wahl hängt von den speziellen Anforderungen Ihres Anwendungsfalls ab. Um mehr über Unterschiede zu erfahren, betrachten Sie die lokal entwickelten Javadocs oder den Quellcode. In den folgenden Abschnitten wird die Mindestimplementierung für die ersten Schritte beschrieben.

IRecordProzessorversionen

- Ursprüngliche Schnittstelle (Version 1)
- Aktualisierte Schnittstelle (Version 2)

Ursprüngliche Schnittstelle (Version 1)

Die ursprüngliche `IRecordProcessor` Schnittstelle (`package com.amazonaws.services.kinesis.clientlibrary.interfaces`) stellt die folgenden Datensatzverarbeitermethoden bereit, die Ihr Konsument implementieren muss. Das Beispiel stellt Implementierungen bereit, die Sie als Ausgangspunkt verwenden können (siehe `AmazonKinesisApplicationSampleRecordProcessor.java`).

```
public void initialize(String shardId)
public void processRecords(List<Record> records, IRecordProcessorCheckpointer
 checkpointer)
public void shutdown(IRecordProcessorCheckpointer checkpointer, ShutdownReason reason)
```

initialize

Die KCL ruft die Methode `initialize` auf, wenn der Datensatzverarbeiter instanziiert wird, und übergibt eine spezifische Shard-ID als Parameter. Dieser Datensatzverarbeiter verarbeitet nur diese Shard und in der Regel ist dies auch umgekehrt der Fall (diese Shard wird nur durch diesen Datensatverarbeiter verarbeitet). Ihr Konsument sollte jedoch die Möglichkeit berücksichtigen, dass ein Datensatz mehr als einmal verarbeitet werden könnte. Kinesis Data Streams besitzt eine

Semantik nach dem Grundsatz mindestens einmal. Das bedeutet, dass jeder Datensatz aus einer Shard mindestens einmal von einem Worker in Ihrem Konsumenten verarbeitet wird. Weitere Informationen zu Fällen, in denen eine bestimmte Shard möglicherweise durch mehr als einen Auftragnehmer verarbeitet wird, finden Sie unter Verwenden Sie Resharding, Skalierung und Parallelverarbeitung, um die Anzahl der Shards zu ändern.

```
public void initialize(String shardId)
```

processRecords

Die KCL ruft die Methode `processRecords` auf und übergibt eine Liste der Datensätze aus der Shard, die von der Methode `initialize(shardId)` angegeben wird. Der Datensatzverarbeiter verarbeitet die Daten in diesen Datensätzen entsprechend der Semantik des Konsumenten. Beispielsweise kann der Auftragnehmer eine Transformation für die Daten ausführen und das Ergebnis dann in einem Amazon Simple Storage Service (Amazon S3)-Bucket speichern.

```
public void processRecords(List<Record> records, IRecordProcessorCheckpointer
 checkpointer)
```

Zusätzlich zu den Daten selbst enthält der Datensatz auch eine Sequenznummer und einen Partitionsschlüssel. Der Auftragnehmer kann diese Werte beim Verarbeiten der Daten verwenden. Beispielsweise könnte der Auftragnehmer basierend auf dem Wert des Partitionsschlüssels den S3-Bucket wählen, in dem die Daten gespeichert werden sollen. Die Klasse `Record` stellt die folgenden Methoden bereit, die Zugriff auf die Daten des Datensatzes, die Sequenznummer und den Partitionsschlüssel bieten.

```
record.getData()
record.getSequenceNumber()
record.getPartitionKey()
```

In diesem Beispiel weist die private Methode `processRecordsWithRetries` Code auf, der zeigt, wie ein Auftragnehmer auf die Daten des Datensatzes, die Sequenznummer und den Partitionsschlüssel zugreifen kann.

Kinesis Data Streams erfordert, dass der Datensatzverarbeiter die Datensätze nachverfolgt, die bereits in einer Shard verarbeitet wurden. Die KCL übernimmt diese Nachverfolgung für Sie, indem ein Checkpointer (`IRecordProcessorCheckpointer`) an `processRecords` übergeben wird. Der Datensatzverarbeiter ruft die Methode `checkpoint` auf dieser Schnittstelle auf, um die KCL über die Fortschritte zu informieren, die sie beim Verarbeiten der Datensätze in der Shard gemacht hat.

Wenn der Auftragnehmer fehlschlägt, verwendet die KCL diese Informationen, um die Verarbeitung der Shard mit dem letzten bekannten Datensatz neu zu starten.

Im Fall einer Teilungs- oder Zusammenführungsoperation beginnt die KCL erst dann mit der Verarbeitung der neuen Shards, wenn die Verarbeiter für die ursprünglichen Shards `checkpoint` aufgerufen haben, um zu signalisieren, dass die Verarbeitung der ursprünglichen Shards vollständig abgeschlossen ist.

Wenn Sie keinen Parameter übergeben, nimmt die KCL an, dass der Aufruf von `checkpoint` bedeutet, dass alle Datensätze bis zum letzten Datensatz, der an den Datensatzverarbeiter übergeben wurde, verarbeitet wurden. Daher sollte der Datensatzverarbeiter die Methode `checkpoint` erst aufrufen, wenn er alle Datensätze in der Liste, die ihm übergeben wurden, verarbeitet hat. Datensatzverarbeiter müssen `checkpoint` nicht bei jedem Aufruf von `processRecords` aufrufen. Ein Prozessor könnte beispielsweise `checkpoint` bei jedem dritten Aufruf von `processRecords` aufrufen. Sie können optional die exakte Sequenznummer eines Datensatzes als Parameter für `checkpoint` angeben. In diesem Fall nimmt die KCL an, dass alle Datensätze nur bis zu diesem Datensatz verarbeitet wurden.

Im Beispiel zeigt die private Methode `checkpoint`, wie `IRecordProcessorCheckpointer.checkpoint` mithilfe der entsprechenden Ausnahmebehandlung und Wiederholungslogik aufgerufen wird.

Die KCL ist bei der Behandlung von Ausnahmen, die während der Verarbeitung der Datensätze auftreten, von `processRecords` abhängig. Wenn `processRecords` eine Ausnahme aufwirft, überspringt die KCL die Datensätze, die vor der Ausnahme übergeben wurden. Das heißt, diese Datensätze werden nicht erneut an den Datensatzprozessor gesendet, der die Ausnahme ausgelöst hat, oder an einen anderen Datensatzprozessor im Verbraucher.

shutdown

Die KCL ruft die Methode `shutdown` entweder auf, wenn die Verarbeitung beendet wird (Grund für das Herunterfahren ist `TERMINATE`) oder wenn der Auftragnehmer nicht mehr reagiert (Grund für das Herunterfahren ist `ZOMBIE`).

```
public void shutdown(IRecordProcessorCheckpointer checkpointer, ShutdownReason reason)
```

Die Verarbeitung endet, wenn der Datensatzverarbeiter keine weiteren Datensätze aus der Shard erhält, entweder weil die Shard geteilt oder zusammengeführt wurde oder weil der Stream gelöscht wurde.

Die KCL übergibt auch eine `IRecordProcessorCheckpointer`-Schnittstelle an `shutdown`. Wenn der Grund für das Herunterfahren `TERMINATE` ist, sollte der Datensatzverarbeiter alle Datensätze fertigstellen und dann die Methode `checkpoint` in seiner Schnittstelle aufrufen.

Aktualisierte Schnittstelle (Version 2)

Die aktualisierte `IRecordProcessor` Schnittstelle (`package com.amazonaws.services.kinesis.clientlibrary.interfaces.v2`) stellt die folgenden Datensatzverarbeitermethoden bereit, die Ihr Konsument implementieren muss:

```
void initialize(InitializationInput initializationInput)
void processRecords(ProcessRecordsInput processRecordsInput)
void shutdown(ShutdownInput shutdownInput)
```

Sie können auf alle Argumente aus der ursprünglichen Version der Schnittstelle über Get-Methoden für die Container-Objekte zugreifen. Um die Liste der Datensätze in `processRecords()` abzurufen, können Sie `processRecordsInput.getRecords()` verwenden.

Ab Version 2 dieser Schnittstelle (KCL 1.5.0 und höher) sind zusätzlich zu den Eingaben durch die ursprüngliche Schnittstelle die folgenden neuen Eingaben verfügbar:

Startsequenznummer

Im `InitializationInput`-Objekt, das an die Operation `initialize()` übergeben wird, die Startsequenznummer, aus der Datensätze für die Datenverarbeiter-Instance bereitgestellt würden. Dies ist die Sequenznummer, die zuletzt durch die Datensatzverarbeiter-Instance überprüft wurde, die dieselbe Shard zuvor verarbeitet hat. Sie wird für den Fall angegeben, dass Ihre Anwendung diese Informationen benötigt.

Ausstehende Checkpoint-Sequenznummer

Im `InitializationInput`-Objekt, das an die Operation `initialize()` übergeben wird, die ausstehende Checkpoint-Sequenznummer (wenn vorhanden), die nicht übergeben werden konnte, bevor die vorherige Datensatzverarbeiter-Instance angehalten wurde.

Implementieren Sie eine Klassenfabrik für die IRecord Prozessorschnittstelle

Sie müssen darüber hinaus eine Factory für die Klasse implementieren, die die Datensatzverarbeitermethoden implementiert. Wenn der Konsument den Auftragnehmer instanziiert, übergibt er dieser Factory eine Referenz.

Im Beispiel wird die Factory-Klasse in der Datei `AmazonKinesisApplicationSampleRecordProcessorFactory.java` mithilfe der ursprünglichen Datensatzverarbeiter-Schnittstelle implementiert. Wenn Sie möchten, dass die Class Factory Datensatzverarbeiter mit Version 2 erstellt, verwenden Sie den Paketnamen `com.amazonaws.services.kinesis.clientlibrary.interfaces.v2`.

```
public class SampleRecordProcessorFactory implements IRecordProcessorFactory {
    /**
    * Constructor.
    */
    public SampleRecordProcessorFactory() {
        super();
    }
    /**
    * {@inheritDoc}
    */
    @Override
    public IRecordProcessor createProcessor() {
        return new SampleRecordProcessor();
    }
}
```

Erstellen Sie einen Worker

Wie in Implementieren Sie die Methoden des Prozessors IRecord beschrieben, gibt es zwei Versionen der KCL-Datensatzverarbeiterschnittstelle zur Auswahl. Die Version hat Auswirkungen auf die Art, wie Sie einen Worker erstellen. Die ursprüngliche Datensatzverarbeiterschnittstelle verwendet die folgende Codestruktur, um einen Auftragnehmer zu erstellen:

```
final KinesisClientLibConfiguration config = new KinesisClientLibConfiguration(...)
final IRecordProcessorFactory recordProcessorFactory = new RecordProcessorFactory();
final Worker worker = new Worker(recordProcessorFactory, config);
```

Mit Version 2 der Datensatzverarbeiterschnittstelle können Sie `Worker.Builder` verwenden, um einen Auftragnehmer zu erstellen, ohne sich Gedanken über den Konstruktor und die Reihenfolge der Argumente zu machen. Die aktualisierte Datensatzverarbeiterschnittstelle verwendet die folgende Codestruktur, um einen Auftragnehmer zu erstellen:

```
final KinesisClientLibConfiguration config = new KinesisClientLibConfiguration(...)
```

```
final IRecordProcessorFactory recordProcessorFactory = new RecordProcessorFactory();
final Worker worker = new Worker.Builder()
    .recordProcessorFactory(recordProcessorFactory)
    .config(config)
    .build();
```

Ändern Sie die Konfigurationseigenschaften

Das Beispiel zeigt Standardwerte für Konfigurationseigenschaften. Diese Konfigurationsdaten für den Auftragnehmer werden anschließend in einem `KinesisClientLibConfiguration`-Objekt konsolidiert. Dieses Objekt und eine Referenz auf die Class Factory für `IRecordProcessor` werden an den Aufruf übergeben, der den Auftragnehmer instanziiert. Sie können diese Eigenschaften mithilfe einer Java-Eigenschaftendatei (siehe `AmazonKinesisApplicationSample.java`) durch eigene Werte überschreiben.

Anwendungsname

Die KCL erfordert einen Anwendungsnamen, der unter Ihren Anwendungen sowie den Amazon-DynamoDB-Tabellen in derselben Region eindeutig ist. Sie verwendet den Wert der Anwendungsnamenkonfiguration auf folgende Arten:

- Für mit diesem Anwendungsnamen verknüpfte Auftragnehmer wird angenommen, dass sie gemeinsam im gleichen Stream arbeiten. Diese Auftragnehmer können auf mehrere Instances verteilt sein. Wenn Sie eine zusätzliche Instance desselben Anwendungscodes ausführen, jedoch mit einem anderen Anwendungsnamen, behandelt die KCL die zweite Instance als eine völlig getrennte Anwendung, die ebenfalls im selben Stream arbeitet.
- Die KCL erstellt eine DynamoDB-Tabelle mit dem Namen der Anwendung und verwendet die Tabelle für die Verwaltung von Statusinformationen für die Anwendung (wie Checkpoints und Auftragnehmer-Shard-Zuweisungen). Jede Anwendung verfügt über eine eigene DynamoDB-Tabelle. Weitere Informationen finden Sie unter Verwenden Sie eine Leasetabelle, um nachzuverfolgen, welche Shards von der KCL-Consumer-Anwendung verarbeitet wurden.

Richten Sie Anmeldeinformationen ein

Sie müssen Ihre AWS Anmeldeinformationen einem der Anmeldeinformationsanbieter in der Kette der Standardanmeldedienstanbieter zur Verfügung stellen. Wenn Sie beispielsweise Ihren Konsumenten auf einer EC2-Instance ausführen, empfehlen wir, die Instance mit einer IAM-Rolle zu starten. AWS -Anmeldeinformationen, die die mit dieser IAM-Rolle verknüpften Berechtigungen widerspiegeln, werden den Anwendungen auf der Instance über deren Instance-Metadaten zur

Verfügung gestellt. Dies ist die sicherste Art, Anmeldeinformationen für einen Konsumenten zu verwalten, der auf einer EC2-Instance ausgeführt wird.

Die Beispielanwendung versucht zunächst, IAM-Anmeldeinformationen aus den Instance-Metadaten abzurufen:

```
credentialsProvider = new InstanceProfileCredentialsProvider();
```

Wenn die Beispielanwendung keine Anmeldeinformationen aus den Instance-Metadaten abrufen kann, versucht sie, Anmeldeinformationen aus einer Eigenschaftendatei abzurufen:

```
credentialsProvider = new ClasspathPropertiesFileCredentialsProvider();
```

Weitere Informationen zu Instance-Metadaten finden Sie unter Instance-Metadaten im Amazon EC2 EC2-Benutzerhandbuch.

Verwenden Sie die Worker-ID für mehrere Instances

Derselbe Initialisierungscode erstellt unter Verwendung des Namens des lokalen Computers und Anfügung eines global eindeutigen Bezeichners eine ID für den Auftragnehmer, `workerId`, wie im folgenden Codeauszug gezeigt. Dieser Ansatz unterstützt das Szenario mit mehreren Instances der Konsumentenanwendung, die auf einem einzigen Computer ausgeführt werden.

```
String workerId = InetAddress.getLocalHost().getCanonicalHostName() + ":" +
 UUID.randomUUID();
```

Migrieren Sie zu Version 2 der Record Processor-Schnittstelle

Wenn Sie Code migrieren möchten, der die ursprüngliche Schnittstelle verwendet, sind zusätzlich zu den zuvor beschriebenen Schritten die folgenden Schritte erforderlich:

1. Ändern der Datensatzverarbeiterklasse, um Version 2 der Datensatzverarbeiterschnittstelle zu importieren:

   ```
   import com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
   ```

2. Ändern der Referenzen zu Eingaben, um `get`-Methoden für die Container-Objekte zu verwenden. In der Operation `shutdown()` ändern Sie beispielsweise „`checkpointer`“ in „`shutdownInput.getCheckpointer()`“.

3. Ändern der Datensatzverarbeiter-Factory, um Version 2 der Datensatzverarbeiter-Factory zu importieren:

```
import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFactory;
```

4. Ändern der Konstruktion für den Auftragnehmer, um `Worker.Builder` zu verwenden. Beispiel:

```
final Worker worker = new Worker.Builder()
    .recordProcessorFactory(recordProcessorFactory)
    .config(config)
    .build();
```

Entwickeln Sie einen Kinesis Client Library-Consumer in Node.js

> **Important**
>
> Die Versionen 1.x und 2.x der Amazon Kinesis Client Library (KCL) sind veraltet. KCL 1.x wird end-of-support am 30. Januar 2026 verfügbar sein. Wir empfehlen dringend, dass Sie Ihre KCL-Anwendungen, die Version 1.x verwenden, vor dem 30. Januar 2026 auf die neueste KCL-Version migrieren. Die neueste KCL-Version finden Sie auf der Seite Amazon Kinesis Client Library unter. GitHub Informationen zu den neuesten KCL-Versionen finden Sie unter. Verwenden Sie die Kinesis-Clientbibliothek Informationen zur Migration von KCL 1.x zu KCL 3.x finden Sie unter. Migrieren von KCL 1.x zu KCL 3.x

Sie können die Kinesis Client Library (KCL) verwenden, um Anwendungen zu erstellen, die Daten aus Ihren Kinesis-Datenströmen verarbeiten. Die Kinesis Client Library ist in mehreren Sprachen verfügbar. In diesem Thema wird Node.js behandelt.

Die KCL ist eine Java-Bibliothek. Unterstützung für andere Sprachen als Java wird über eine mehrsprachige Schnittstelle bereitgestellt, die als. MultiLangDaemon Dieser Daemon basiert auf Java und wird im Hintergrund ausgeführt, wenn Sie eine andere KCL-Sprache als Java verwenden. Wenn Sie also die KCL für Node.js installieren und Ihre App für Privatanwender vollständig in Node.js schreiben, müssen Sie trotzdem Java auf Ihrem System installieren, und zwar aufgrund der. MultiLangDaemon Darüber hinaus MultiLangDaemon verfügt es über einige Standardeinstellungen, die Sie möglicherweise an Ihren Anwendungsfall anpassen müssen, z. B. die AWS Region, mit

der eine Verbindung hergestellt wird. Weitere Informationen dazu finden Sie MultiLangDaemon auf GitHub der MultiLangDaemon KCL-Projektseite.

Um die Node.js KCL von herunterzuladen GitHub, gehen Sie zur Kinesis Client Library (Node.js).

Downloads von Beispiel-Code

Es gibt zwei Code-Beispiele für die KCL in Node.js:

- basic-sample

  Wird in den folgenden Abschnitten verwendet, um die Grundlagen zum Erstellen einer KCL-Konsumentenanwendung in Node.js zu zeigen.
- click-stream-sample

  Etwas komplexere und verwendet ein reales Szenario, nachdem Sie sich mit dem grundlegenden Beispiel-Code vertraut gemacht haben. Dieses Beispiel wird hier nicht behandelt. Es gibt jedoch eine Readme-Datei mit weiteren Informationen.

Sie müssen die folgenden Aufgaben durchführen, wenn Sie eine KCL-Konsumentenanwendung in Node.js implementieren:

Aufgaben

- Implementieren Sie den Record Processor
- Ändern Sie die Konfigurationseigenschaften

Implementieren Sie den Record Processor

Der einfachste Konsument, der die KCL für Node.js verwenden kann, muss die Funktion `recordProcessor` implementieren. Diese enthält wiederum die Funktionen `initialize`, `processRecords`, und `shutdown`. Das Beispiel zeigt eine Implementierung, die Sie als Ausgangspunkt verwenden können (siehe `sample_kcl_app.js`).

```
function recordProcessor() {
  // return an object that implements initialize, processRecords and shutdown
 functions.}
```

initialize

Die KCL ruft die Funktion `initialize` auf, wenn der Datensatzverarbeiter gestartet wird. Dieser Datensatzverarbeiter verarbeitet nur die Shard-ID, die als `initializeInput.shardId` übergeben wird. In der Regel ist dies auch umgekehrt der Fall (diese Shard wird nur durch diesen Datensatverarbeiter verarbeitet). Ihr Konsument sollte jedoch die Möglichkeit berücksichtigen, dass ein Datensatz mehr als einmal verarbeitet werden könnte. Das liegt daran, dass Kinesis Data Streams eine Semantik nach dem Grundsatz mindestens einmal hat. Das bedeutet, dass jeder Datensatz aus einer Shard mindestens einmal von einem Worker in Ihrem Konsumenten verarbeitet wird. Weitere Informationen zu Fällen, in denen eine bestimmte Shard möglicherweise durch mehr als einen Auftragnehmer verarbeitet wird, finden Sie unter Verwenden Sie Resharding, Skalierung und Parallelverarbeitung, um die Anzahl der Shards zu ändern.

```
initialize: function(initializeInput, completeCallback)
```

processRecords

Die KCL ruft diese Funktion mit einer Eingabe auf, die eine Liste von Datensätzen aus der für die Funktion `initialize` angegebenen Shard enthält. Der von Ihnen implementierte Datensatzverarbeiter verarbeitet die Daten in diesen Datensätzen entsprechend der Semantik Ihres Konsumenten. Beispielsweise kann der Auftragnehmer eine Transformation für die Daten ausführen und das Ergebnis dann in einem Amazon Simple Storage Service (Amazon S3)-Bucket speichern.

```
processRecords: function(processRecordsInput, completeCallback)
```

Zusätzlich zu den Daten enthält der Datensatz auch eine Sequenznummer und einen Partitionsschlüssel, die der Auftragnehmer beim Verarbeiten der Daten verwenden kann. Beispielsweise könnte der Auftragnehmer basierend auf dem Wert des Partitionsschlüssels den S3-Bucket wählen, in dem die Daten gespeichert werden sollen. Das `record`-Anmeldeverzeichnis stellt die folgenden Schlüssel-Wert-Paare für den Zugriff auf die Daten des Datensatzes, die Sequenznummer und den Partitionsschlüssel bereit:

```
record.data
record.sequenceNumber
record.partitionKey
```

Beachten Sie, dass die Daten Base64-kodiert sind.

Im einfachen Beispiel weist die Funktion `processRecords` Code auf, der zeigt, wie ein Auftragnehmer auf die Daten des Datensatzes, die Sequenznummer und den Partitionsschlüssel zugreifen kann.

Kinesis Data Streams erfordert, dass der Datensatzverarbeiter die Datensätze nachverfolgt, die bereits in einer Shard verarbeitet wurden. Die KCL übernimmt die Nachverfolgung durch ein `checkpointer`-Objekt, das als `processRecordsInput.checkpointer` übergeben wird. Der Datensatzverarbeiter ruft die Funktion `checkpointer.checkpoint` auf, um die KCL über die Fortschritte zu informieren, die er beim Verarbeiten der Datensätze in der Shard gemacht hat. Wenn der Auftragnehmer fehlschlägt, verwendet die KCL diese Informationen, wenn Sie die Verarbeitung der Shard erneut starten, damit sie den Vorgang ab dem letzten bekannten Datensatz fortsetzt.

Im Fall einer Teilungs- oder Zusammenführungsoperation beginnt die KCL erst dann mit der Verarbeitung der neuen Shards, wenn die Verarbeiter für die ursprünglichen Shards `checkpoint` aufgerufen haben, um zu signalisieren, dass die Verarbeitung der ursprünglichen Shards vollständig abgeschlossen ist.

Wenn Sie die Sequenznummer nicht an die `checkpoint`-Funktion übergeben, nimmt die KCL an, dass der Aufruf von `checkpoint` bedeutet, dass alle Datensätze bis zum letzten Datensatz, der an den Datensatzverarbeiter übergeben wurde, verarbeitet wurden. Daher sollte der Datensatzverarbeiter die Methode `checkpoint` erst aufrufen, wenn er alle Datensätze in der Liste, die ihm übergeben wurden, verarbeitet hat. Datensatzverarbeiter müssen `checkpoint` nicht bei jedem Aufruf von `processRecords` aufrufen. Ein Prozessor könnte beispielsweise bei jedem dritten Anruf oder `checkpoint` bei einem Ereignis außerhalb Ihres Aufzeichnungsprozessors, z. B. bei einem benutzerdefinierten verification/validation Dienst, den Sie implementiert haben, aufrufen.

Sie können optional die exakte Sequenznummer eines Datensatzes als Parameter für `checkpoint` angeben. In diesem Fall nimmt die KCL an, dass alle Datensätze nur bis zu diesem Datensatz verarbeitet wurden.

Die einfache Beispielanwendung zeigt den einfachsten möglichen Aufruf der Funktion `checkpointer.checkpoint`. Sie können weitere Checkpoint-Logik hinzufügen, die Sie an diesem Punkt in der Funktion für Ihren Konsumenten benötigen.

shutdown

Die KCL ruft die Funktion `shutdown` entweder auf, wenn die Verarbeitung beendet wird (`shutdownInput.reason` ist `TERMINATE`) oder wenn der Auftragnehmer nicht mehr reagiert (`shutdownInput.reason` ist `ZOMBIE`).

```
shutdown: function(shutdownInput, completeCallback)
```

Die Verarbeitung endet, wenn der Datensatzverarbeiter keine weiteren Datensätze aus der Shard erhält, entweder weil die Shard geteilt oder zusammengeführt wurde oder weil der Stream gelöscht wurde.

Die KCL übergibt auch ein `shutdownInput.checkpointer`-Objekt an `shutdown`. Wenn der Grund für das Herunterfahren `TERMINATE` ist, sollten Sie sicherstellen, dass der Datensatzverarbeiter die Verarbeitung aller Datensätze fertiggestellt hat, und dann die Funktion `checkpoint` in seiner Schnittstelle aufrufen.

Ändern Sie die Konfigurationseigenschaften

Das Beispiel zeigt Standardwerte für die Konfigurationseigenschaften. Sie können diese Eigenschaften mit eigenen Werten überschreiben (siehe `sample.properties` im einfachen Beispiel).

Anwendungsname

Die KCL erfordert eine Anwendung, die unter Ihren Anwendungen sowie den Amazon-DynamoDB-Tabellen in derselben Region eindeutig ist. Sie verwendet den Wert der Anwendungsnamenkonfiguration auf folgende Arten:

- Für mit diesem Anwendungsnamen verknüpfte Auftragnehmer wird angenommen, dass sie gemeinsam im gleichen Stream arbeiten. Diese Auftragnehmer können auf mehrere Instances verteilt sein. Wenn Sie eine zusätzliche Instance desselben Anwendungscodes ausführen, jedoch mit einem anderen Anwendungsnamen, behandelt die KCL die zweite Instance als eine völlig getrennte Anwendung, die ebenfalls im selben Stream arbeitet.
- Die KCL erstellt eine DynamoDB-Tabelle mit dem Namen der Anwendung und verwendet die Tabelle für die Verwaltung von Statusinformationen für die Anwendung (wie Checkpoints und Auftragnehmer-Shard-Zuweisungen). Jede Anwendung verfügt über eine eigene DynamoDB-Tabelle. Weitere Informationen finden Sie unter Verwenden Sie eine Leasetabelle, um nachzuverfolgen, welche Shards von der KCL-Consumer-Anwendung verarbeitet wurden.

Richten Sie Anmeldeinformationen ein

Sie müssen Ihre AWS Anmeldeinformationen einem der Anmeldeinformationsanbieter in der Kette der Standardanmeldedienstanbieter zur Verfügung stellen. Sie können die Eigenschaft `AWSCredentialsProvider` verwenden, um einen Anmeldeinformationsanbieter einzurichten. Die `sample.properties`-Datei muss Anmeldeinformationen einem der Anmeldeinformationsanbieter in der Anmeldeinformationsanbieter-Standardkette bereitstellen. Wenn Sie Ihren Consumer auf einer

Amazon EC2 EC2-Instance ausführen, empfehlen wir Ihnen, die Instance mit einer IAM-Rolle zu konfigurieren. AWS Anmeldeinformationen, die die mit dieser IAM-Rolle verknüpften Berechtigungen widerspiegeln, werden Anwendungen auf der Instance über deren Instance-Metadaten zur Verfügung gestellt. Dies ist die sicherste Art, Anmeldeinformationen für eine Konsumentenanwendung zu verwalten, die auf einer EC2-Instance ausgeführt wird.

Im folgenden Beispiel wird eine KCL konfiguriert, um einen Kinesis-Datenstrom namens `kclnodejssample` mittels des Datensatzverarbeiters zu verarbeiten, der in `sample_kcl_app.js` bereitgestellt wird:

```
# The Node.js executable script
executableName = node sample_kcl_app.js
# The name of an Amazon Kinesis stream to process
streamName = kclnodejssample
# Unique KCL application name
applicationName = kclnodejssample
# Use default AWS credentials provider chain
AWSCredentialsProvider = DefaultAWSCredentialsProviderChain
# Read from the beginning of the stream
initialPositionInStream = TRIM_HORIZON
```

Entwickeln Sie einen Kinesis Client Library-Consumer in .NET

> **Important**
>
> Die Versionen 1.x und 2.x der Amazon Kinesis Client Library (KCL) sind veraltet. KCL 1.x wird end-of-support am 30. Januar 2026 verfügbar sein. Wir empfehlen dringend, dass Sie Ihre KCL-Anwendungen, die Version 1.x verwenden, vor dem 30. Januar 2026 auf die neueste KCL-Version migrieren. Die neueste KCL-Version finden Sie auf der Seite Amazon Kinesis Client Library unter. GitHub Informationen zu den neuesten KCL-Versionen finden Sie unter. Verwenden Sie die Kinesis-Clientbibliothek Informationen zur Migration von KCL 1.x zu KCL 3.x finden Sie unter. Migrieren von KCL 1.x zu KCL 3.x

Sie können die Kinesis Client Library (KCL) verwenden, um Anwendungen zu erstellen, die Daten aus Ihren Kinesis-Datenströmen verarbeiten. Die Kinesis Client Library ist in mehreren Sprachen verfügbar. In diesem Thema wird .NET behandelt.

Die KCL ist eine Java-Bibliothek. Unterstützung für andere Sprachen als Java wird über eine mehrsprachige Schnittstelle bereitgestellt, die als. MultiLangDaemon Dieser Daemon basiert auf

Java und wird im Hintergrund ausgeführt, wenn Sie eine andere KCL-Sprache als Java verwenden. Wenn Sie also die KCL für .NET installieren und Ihre Consumer-App vollständig in .NET schreiben, müssen Sie trotzdem Java auf Ihrem System installieren, da MultiLangDaemon Darüber hinaus MultiLangDaemon verfügt es über einige Standardeinstellungen, die Sie möglicherweise an Ihren Anwendungsfall anpassen müssen, z. B. die AWS Region, mit der eine Verbindung hergestellt wird. Weitere Informationen dazu finden Sie MultiLangDaemon auf GitHub der MultiLangDaemon KCL-Projektseite.

Gehen Sie zur Kinesis Client Library (.NET) GitHub, um die.NET-KCL von herunterzuladen. Um Beispielcode für eine.NET-KCL-Consumer-Anwendung herunterzuladen, rufen Sie die Projektseite KCL for .NET-Beispielanwendung für Privatanwender auf. GitHub

Sie müssen die folgenden Aufgaben durchführen, wenn Sie eine KCL-Konsumentenanwendung in .NET implementieren:

Aufgaben

- Implementieren Sie die Methoden der IRecord Prozessorklasse
- Ändern Sie die Konfigurationseigenschaften

Implementieren Sie die Methoden der IRecord Prozessorklasse

Der Konsument muss die folgenden Methoden für `IRecordProcessor` implementieren. Der Konsument im Beispiel stellt Implementierungen bereit, die Sie als Ausgangspunkt verwenden können (siehe die `SampleRecordProcessor`-Klasse in `SampleConsumer/AmazonKinesisSampleConsumer.cs`).

```
public void Initialize(InitializationInput input)
public void ProcessRecords(ProcessRecordsInput input)
public void Shutdown(ShutdownInput input)
```

Initialisieren

Die KCL ruft diese Methode auf, wenn der Datensatzverarbeiter instanziiert wird, und übergibt eine spezifische Shard-ID an den `input`-Parameter (`input.ShardId`). Dieser Datensatzverarbeiter verarbeitet nur diese Shard und in der Regel ist dies auch umgekehrt der Fall (diese Shard wird nur durch diesen Datensatverarbeiter verarbeitet). Ihr Konsument sollte jedoch die Möglichkeit berücksichtigen, dass ein Datensatz mehr als einmal verarbeitet werden könnte. Das liegt daran, dass Kinesis Data Streams eine Semantik nach dem Grundsatz mindestens einmal hat. Das bedeutet, dass jeder Datensatz aus einer Shard mindestens einmal von einem Worker in Ihrem

Konsumenten verarbeitet wird. Weitere Informationen zu Fällen, in denen eine bestimmte Shard möglicherweise durch mehr als einen Auftragnehmer verarbeitet wird, finden Sie unter Verwenden Sie Resharding, Skalierung und Parallelverarbeitung, um die Anzahl der Shards zu ändern.

```
public void Initialize(InitializationInput input)
```

ProcessRecords

Die KCL ruft diese Methode auf und übergibt eine Liste der Datensätze an den `input`-Parameter (`input.Records`) aus der Shard, die von der Methode `Initialize` angegeben wird. Der von Ihnen implementierte Datensatzverarbeiter verarbeitet die Daten in diesen Datensätzen entsprechend der Semantik Ihres Konsumenten. Beispielsweise kann der Auftragnehmer eine Transformation für die Daten ausführen und das Ergebnis dann in einem Amazon Simple Storage Service (Amazon S3)-Bucket speichern.

```
public void ProcessRecords(ProcessRecordsInput input)
```

Zusätzlich zu den Daten selbst enthält der Datensatz auch eine Sequenznummer und einen Partitionsschlüssel. Der Auftragnehmer kann diese Werte beim Verarbeiten der Daten verwenden. Beispielsweise könnte der Auftragnehmer basierend auf dem Wert des Partitionsschlüssels den S3-Bucket wählen, in dem die Daten gespeichert werden sollen. Die Klasse `Record` stellt die folgenden Methoden bereit, die Zugriff auf die Daten des Datensatzes, die Sequenznummer und den Partitionsschlüssel bieten:

```
byte[] Record.Data
string Record.SequenceNumber
string Record.PartitionKey
```

Im Beispiel weist die Methode `ProcessRecordsWithRetries` Code auf, der zeigt, wie ein Auftragnehmer auf die Daten des Datensatzes, die Sequenznummer und den Partitionsschlüssel zugreifen kann.

Kinesis Data Streams erfordert, dass der Datensatzverarbeiter die Datensätze nachverfolgt, die bereits in einer Shard verarbeitet wurden. Die KCL übernimmt diese Nachverfolgung für Sie, indem ein `Checkpointer`-Objekt an `ProcessRecords` (`input.Checkpointer`) übergeben wird. Der Datensatzverarbeiter ruft die Methode `Checkpointer.Checkpoint` auf, um die KCL über die Fortschritte zu informieren, die sie beim Verarbeiten der Datensätze in der Shard gemacht hat. Wenn der Auftragnehmer fehlschlägt, verwendet die KCL diese Informationen, um die Verarbeitung der Shard mit dem letzten bekannten Datensatz neu zu starten.

Im Fall einer Teilungs- oder Zusammenführungsoperation beginnt die KCL erst dann mit der Verarbeitung der neuen Shards, wenn die Verarbeiter für die ursprünglichen Shards `Checkpointer.Checkpoint` aufgerufen haben, um zu signalisieren, dass die Verarbeitung der ursprünglichen Shards vollständig abgeschlossen ist.

Wenn Sie keinen Parameter übergeben, nimmt die KCL an, dass der Aufruf von `Checkpointer.Checkpoint` bedeutet, dass alle Datensätze bis zum letzten Datensatz, der an den Datensatzverarbeiter übergeben wurde, verarbeitet wurden. Daher sollte der Datensatzverarbeiter die Methode `Checkpointer.Checkpoint` erst aufrufen, wenn er alle Datensätze in der Liste, die ihm übergeben wurden, verarbeitet hat. Datensatzverarbeiter müssen `Checkpointer.Checkpoint` nicht bei jedem Aufruf von `ProcessRecords` aufrufen. Ein Prozessor könnte beispielsweise `Checkpointer.Checkpoint` bei jedem dritten oder vierten Aufruf aufrufen. Sie können optional die exakte Sequenznummer eines Datensatzes als Parameter für `Checkpointer.Checkpoint` angeben. In diesem Fall nimmt die KCL an, dass die Datensätze nur bis zu diesem Datensatz verarbeitet wurden.

Im Beispiel zeigt die private Methode `Checkpoint(Checkpointer checkpointer)`, wie die `Checkpointer.Checkpoint`-Methode mithilfe der entsprechenden Ausnahmebehandlung und Wiederholungslogik aufgerufen wird.

Die KCL für .NET verarbeitet Ausnahmen anders als andere KCL-Sprachbibliotheken, da sie keine Ausnahmen verarbeitet, die aus der Verarbeitung der Datensätze entstanden sind. Alle nicht abgefangenen Ausnahmen vom Benutzer-Code bringen das Programm zum Absturz.

Herunterfahren

Die KPL ruft die Methode `Shutdown` entweder auf, wenn die Verarbeitung beendet wird (Grund für das Herunterfahren ist `TERMINATE`) oder wenn der Auftragnehmer nicht mehr reagiert (der `input.Reason`-Wert für das Herunterfahren ist `ZOMBIE`).

```
public void Shutdown(ShutdownInput input)
```

Die Verarbeitung endet, wenn der Datensatzverarbeiter keine weiteren Datensätze aus der Shard erhält, weil die Shard geteilt oder zusammengeführt wurde oder der Stream gelöscht wurde.

Die KCL übergibt auch ein `Checkpointer`-Objekt an `shutdown`. Wenn der Grund für das Herunterfahren `TERMINATE` ist, sollte der Datensatzverarbeiter alle Datensätze fertigstellen und dann die Methode `checkpoint` in seiner Schnittstelle aufrufen.

Ändern Sie die Konfigurationseigenschaften

Der Beispielkonsument zeigt Standardwerte für die Konfigurationseigenschaften. Sie können diese Eigenschaften mit eigenen Werten überschreiben (siehe `SampleConsumer/kcl.properties`).

Anwendungsname

Die KCL erfordert eine Anwendung, die unter Ihren Anwendungen sowie den Amazon-DynamoDB-Tabellen in derselben Region eindeutig ist. Sie verwendet den Wert der Anwendungsnamenkonfiguration auf folgende Arten:

- Für mit diesem Anwendungsnamen verknüpfte Auftragnehmer wird angenommen, dass sie gemeinsam im gleichen Stream arbeiten. Diese Auftragnehmer können auf mehrere Instances verteilt sein. Wenn Sie eine zusätzliche Instance desselben Anwendungscodes ausführen, jedoch mit einem anderen Anwendungsnamen, behandelt die KCL die zweite Instance als eine völlig getrennte Anwendung, die ebenfalls im selben Stream arbeitet.
- Die KCL erstellt eine DynamoDB-Tabelle mit dem Namen der Anwendung und verwendet die Tabelle für die Verwaltung von Statusinformationen für die Anwendung (wie Checkpoints und Auftragnehmer-Shard-Zuweisungen). Jede Anwendung verfügt über eine eigene DynamoDB-Tabelle. Weitere Informationen finden Sie unter Verwenden Sie eine Leasetabelle, um nachzuverfolgen, welche Shards von der KCL-Consumer-Anwendung verarbeitet wurden.

Richten Sie Anmeldeinformationen ein

Sie müssen Ihre AWS Anmeldeinformationen einem der Anmeldeinformationsanbieter in der Kette der Standardanmeldedienstanbieter zur Verfügung stellen. Sie können die Eigenschaft `AWSCredentialsProvider` verwenden, um einen Anmeldeinformationsanbieter einzurichten. Die sample.properties muss Ihre Anmeldeinformationen einem der Anmeldeinformationsanbieter in der Anmeldeinformationsanbieter-Standardkette bereitstellen. Wenn Sie Ihre Konsumentenanwendung auf einer EC2-Instance ausführen, empfehlen wir, die Instance mit einer IAM-Rolle zu konfigurieren. AWS -Anmeldeinformationen, die die mit dieser IAM-Rolle verknüpften Berechtigungen widerspiegeln, werden den Anwendungen auf der Instance über deren Instance-Metadaten zur Verfügung gestellt. Dies ist die sicherste Art, Anmeldeinformationen für einen Konsumenten zu verwalten, der auf einer EC2-Instance ausgeführt wird.

Die Eigenschaftendatei des Beispiels konfiguriert KCL, um einen Kinesis-Datenstrom namens „words“ mittels des Datensatzverarbeiters zu verarbeiten, der in `AmazonKinesisSampleConsumer.cs` bereitgestellt wird.

Entwickeln Sie einen Kinesis Client Library-Consumer in Python

> **Important**
>
> Die Versionen 1.x und 2.x der Amazon Kinesis Client Library (KCL) sind veraltet. KCL 1.x wird end-of-support am 30. Januar 2026 verfügbar sein. Wir empfehlen dringend, dass Sie Ihre KCL-Anwendungen, die Version 1.x verwenden, vor dem 30. Januar 2026 auf die neueste KCL-Version migrieren. Die neueste KCL-Version finden Sie auf der Seite Amazon Kinesis Client Library unter. GitHub Informationen zu den neuesten KCL-Versionen finden Sie unter. Verwenden Sie die Kinesis-Clientbibliothek Informationen zur Migration von KCL 1.x zu KCL 3.x finden Sie unter. Migrieren von KCL 1.x zu KCL 3.x

Sie können die Kinesis Client Library (KCL) verwenden, um Anwendungen zu erstellen, die Daten aus Ihren Kinesis-Datenströmen verarbeiten. Die Kinesis Client Library ist in mehreren Sprachen verfügbar. In diesem Thema wird Python behandelt.

Die KCL ist eine Java-Bibliothek. Unterstützung für andere Sprachen als Java wird über eine mehrsprachige Schnittstelle bereitgestellt, die als. MultiLangDaemon Dieser Daemon basiert auf Java und wird im Hintergrund ausgeführt, wenn Sie eine andere KCL-Sprache als Java verwenden. Wenn Sie also die KCL für Python installieren und Ihre Consumer-App vollständig in Python schreiben, muss Java aufgrund der MultiLangDaemon trotzdem auf Ihrem System installiert sein. Darüber hinaus MultiLangDaemon verfügt es über einige Standardeinstellungen, die Sie möglicherweise an Ihren Anwendungsfall anpassen müssen, z. B. die AWS Region, mit der eine Verbindung hergestellt wird. Weitere Informationen dazu finden Sie MultiLangDaemon auf GitHub der MultiLangDaemon KCL-Projektseite.

Um die Python-KCL von herunterzuladen GitHub, gehen Sie zur Kinesis Client Library (Python). Um Beispielcode für eine Python-KCL-Consumer-Anwendung herunterzuladen, gehen Sie zur KCL for Python-Beispielprojektseite unter. GitHub

Sie müssen die folgenden Aufgaben durchführen, wenn Sie eine KCL-Konsumentenanwendung in Python implementieren:

Aufgaben

- Implementieren Sie die Klassenmethoden RecordProcessor
- Ändern Sie die Konfigurationseigenschaften

Implementieren Sie die Klassenmethoden RecordProcessor

Die `RecordProcess`-Klasse muss die `RecordProcessorBase` erweitern, um die folgenden Methoden zu implementieren. Das Beispiel stellt Implementierungen bereit, die Sie als Ausgangspunkt verwenden können (siehe `sample_kclpy_app.py`).

```
def initialize(self, shard_id)
def process_records(self, records, checkpointer)
def shutdown(self, checkpointer, reason)
```

initialize

Die KCL ruft die Methode `initialize` auf, wenn der Datensatzverarbeiter instanziiert wird, und übergibt eine spezifische Shard-ID als Parameter. Dieser Datensatzverarbeiter verarbeitet nur diese Shard und in der Regel ist dies auch umgekehrt der Fall (diese Shard wird nur durch diesen Datensatverarbeiter verarbeitet). Ihr Konsument sollte jedoch die Möglichkeit berücksichtigen, dass ein Datensatz mehr als einmal verarbeitet werden könnte. Das liegt daran, dass Kinesis Data Streams eine Semantik nach dem Grundsatz mindestens einmal hat. Das bedeutet, dass jeder Datensatz aus einer Shard mindestens einmal von einem Worker in Ihrem Konsumenten verarbeitet wird. Weitere Informationen zu Fällen, in denen eine bestimmte Shard möglicherweise durch mehr als einen Auftragnehmer verarbeitet wird, finden Sie unter Verwenden Sie Resharding, Skalierung und Parallelverarbeitung, um die Anzahl der Shards zu ändern.

```
def initialize(self, shard_id)
```

process_records

Die KCL ruft diese Methode auf und übergibt eine Liste der Datensätze aus der Shard, die von der Methode `initialize` angegeben wird. Der von Ihnen implementierte Datensatzverarbeiter verarbeitet die Daten in diesen Datensätzen entsprechend der Semantik Ihres Konsumenten. Beispielsweise kann der Auftragnehmer eine Transformation für die Daten ausführen und das Ergebnis dann in einem Amazon Simple Storage Service (Amazon S3)-Bucket speichern.

```
def process_records(self, records, checkpointer)
```

Zusätzlich zu den Daten selbst enthält der Datensatz auch eine Sequenznummer und einen Partitionsschlüssel. Der Auftragnehmer kann diese Werte beim Verarbeiten der Daten verwenden. Beispielsweise könnte der Auftragnehmer basierend auf dem Wert des Partitionsschlüssels den S3-Bucket wählen, in dem die Daten gespeichert werden sollen. Das `record`-Anmeldeverzeichnis

stellt die folgenden Schlüssel-Wert-Paare für den Zugriff auf die Daten des Datensatzes, die Sequenznummer und den Partitionsschlüssel bereit:

```
record.get('data')
record.get('sequenceNumber')
record.get('partitionKey')
```

Beachten Sie, dass die Daten Base64-kodiert sind.

Im Beispiel weist die Methode `process_records` Code auf, der zeigt, wie ein Auftragnehmer auf die Daten des Datensatzes, die Sequenznummer und den Partitionsschlüssel zugreifen kann.

Kinesis Data Streams erfordert, dass der Datensatzverarbeiter die Datensätze nachverfolgt, die bereits in einer Shard verarbeitet wurden. Die KCL übernimmt diese Nachverfolgung für Sie, indem ein `Checkpointer`-Objekt an `process_records` übergeben wird. Der Datensatzverarbeiter ruft die Methode `checkpoint` auf diesem Objekt auf, um die KCL über die Fortschritte zu informieren, die er beim Verarbeiten der Datensätze in der Shard gemacht hat. Wenn der Auftragnehmer fehlschlägt, verwendet die KCL diese Informationen, um die Verarbeitung der Shard mit dem letzten bekannten Datensatz neu zu starten.

Im Fall einer Teilungs- oder Zusammenführungsoperation beginnt die KCL erst dann mit der Verarbeitung der neuen Shards, wenn die Verarbeiter für die ursprünglichen Shards `checkpoint` aufgerufen haben, um zu signalisieren, dass die Verarbeitung der ursprünglichen Shards vollständig abgeschlossen ist.

Wenn Sie keinen Parameter übergeben, nimmt die KCL an, dass der Aufruf von `checkpoint` bedeutet, dass alle Datensätze bis zum letzten Datensatz, der an den Datensatzverarbeiter übergeben wurde, verarbeitet wurden. Daher sollte der Datensatzverarbeiter die Methode `checkpoint` erst aufrufen, wenn er alle Datensätze in der Liste, die ihm übergeben wurden, verarbeitet hat. Datensatzverarbeiter müssen `checkpoint` nicht bei jedem Aufruf von `process_records` aufrufen. Ein Prozessor könnte beispielsweise `checkpoint` bei jedem dritten Aufruf aufrufen. Sie können optional die exakte Sequenznummer eines Datensatzes als Parameter für `checkpoint` angeben. In diesem Fall nimmt die KCL an, dass alle Datensätze nur bis zu diesem Datensatz verarbeitet wurden.

Im Beispiel zeigt die private Methode `checkpoint`, wie die `Checkpointer.checkpoint`-Methode mithilfe der entsprechenden Ausnahmebehandlung und Wiederholungslogik aufgerufen wird.

Die KCL ist bei der Behandlung von Ausnahmen, die während der Verarbeitung der Datensätze auftreten, von `process_records` abhängig. Wenn `process_records` eine Ausnahme auslöst,

überspringt die KCL die Datensätze, die vor der Ausnahme an `process_records` übergeben wurden. Das heißt, diese Datensätze werden nicht erneut an den Datensatzprozessor gesendet, der die Ausnahme ausgelöst hat, oder an einen anderen Datensatzprozessor im Verbraucher.

shutdown

Die KCL ruft die Methode `shutdown` entweder auf, wenn die Verarbeitung beendet wird (Grund für das Herunterfahren ist `TERMINATE`) oder wenn der Auftragnehmer nicht mehr reagiert (das Herunterfahren `reason` ist `ZOMBIE`).

```
def shutdown(self, checkpointer, reason)
```

Die Verarbeitung endet, wenn der Datensatzverarbeiter keine weiteren Datensätze aus der Shard erhält, entweder weil die Shard geteilt oder zusammengeführt wurde oder weil der Stream gelöscht wurde.

Die KCL übergibt auch ein `Checkpointer`-Objekt an `shutdown`. Wenn der `reason` für das Herunterfahren `TERMINATE` ist, sollte der Datensatzverarbeiter alle Datensätze fertigstellen und dann die Methode `checkpoint` in seiner Schnittstelle aufrufen.

Ändern Sie die Konfigurationseigenschaften

Das Beispiel zeigt Standardwerte für die Konfigurationseigenschaften. Sie können diese Eigenschaften mit eigenen Werten überschreiben (siehe `sample.properties`).

Anwendungsname

Die KCL erfordert einen Anwendungsnamen, der unter Ihren Anwendungen sowie den Amazon-DynamoDB-Tabellen in derselben Region eindeutig ist. Sie verwendet den Wert der Anwendungsnamenkonfiguration auf folgende Arten:

- Für mit diesem Anwendungsnamen verknüpfte Auftragnehmer wird angenommen, dass sie gemeinsam im gleichen Stream arbeiten. Diese Auftragnehmer können auf mehrere Instances verteilt sein. Wenn Sie eine zusätzliche Instance desselben Anwendungscodes ausführen, jedoch mit einem anderen Anwendungsnamen, behandelt die KCL die zweite Instance als eine völlig getrennte Anwendung, die ebenfalls im selben Stream arbeitet.
- Die KCL erstellt eine DynamoDB-Tabelle mit dem Namen der Anwendung und verwendet die Tabelle für die Verwaltung von Statusinformationen für die Anwendung (wie Checkpoints und Auftragnehmer-Shard-Zuweisungen). Jede Anwendung verfügt über eine eigene DynamoDB-

Tabelle. Weitere Informationen finden Sie unter Verwenden Sie eine Leasetabelle, um nachzuverfolgen, welche Shards von der KCL-Consumer-Anwendung verarbeitet wurden.

Richten Sie Anmeldeinformationen ein

Sie müssen Ihre AWS Anmeldeinformationen einem der Anmeldeinformationsanbieter in der Kette der Standardanmeldedienstanbieter zur Verfügung stellen. Sie können die Eigenschaft `AWSCredentialsProvider` verwenden, um einen Anmeldeinformationsanbieter einzurichten. Die sample.properties muss Ihre Anmeldeinformationen einem der Anmeldeinformationsanbieter in der Anmeldeinformationsanbieter-Standardkette bereitstellen. Wenn Sie Ihre Konsumentenanwendung auf einer Amazon-EC2-Instance ausführen, empfehlen wir, die Instance mit einer IAM-Rolle zu konfigurieren. AWS -Anmeldeinformationen, die die mit dieser IAM-Rolle verknüpften Berechtigungen widerspiegeln, werden den Anwendungen auf der Instance über deren Instance-Metadaten zur Verfügung gestellt. Dies ist die sicherste Art, Anmeldeinformationen für eine Konsumentenanwendung zu verwalten, die auf einer EC2-Instance ausgeführt wird.

Die Eigenschaftendatei des Beispiels konfiguriert KCL, um einen Kinesis-Datenstrom namens „words“ mittels des Datensatzverarbeiters zu verarbeiten, der in `sample_kclpy_app.py` bereitgestellt wird.

Entwickeln Sie einen Kinesis Client Library-Consumer in Ruby

> **Important**
>
> Die Versionen 1.x und 2.x der Amazon Kinesis Client Library (KCL) sind veraltet. KCL 1.x wird end-of-support am 30. Januar 2026 verfügbar sein. Wir empfehlen dringend, dass Sie Ihre KCL-Anwendungen, die Version 1.x verwenden, vor dem 30. Januar 2026 auf die neueste KCL-Version migrieren. Die neueste KCL-Version finden Sie auf der Seite Amazon Kinesis Client Library unter. GitHub Informationen zu den neuesten KCL-Versionen finden Sie unter. Verwenden Sie die Kinesis-Clientbibliothek Informationen zur Migration von KCL 1.x zu KCL 3.x finden Sie unter. Migrieren von KCL 1.x zu KCL 3.x

Sie können die Kinesis Client Library (KCL) verwenden, um Anwendungen zu erstellen, die Daten aus Ihren Kinesis-Datenströmen verarbeiten. Die Kinesis Client Library ist in mehreren Sprachen verfügbar. In diesem Thema wird Ruby behandelt.

Die KCL ist eine Java-Bibliothek. Unterstützung für andere Sprachen als Java wird über eine mehrsprachige Schnittstelle bereitgestellt, die als. MultiLangDaemon Dieser Daemon basiert auf Java und wird im Hintergrund ausgeführt, wenn Sie eine andere KCL-Sprache als Java verwenden.

Wenn Sie also die KCL für Ruby installieren und Ihre Consumer-App vollständig in Ruby schreiben, müssen Sie trotzdem Java auf Ihrem System installieren, da MultiLangDaemon Darüber hinaus MultiLangDaemon verfügt es über einige Standardeinstellungen, die Sie möglicherweise an Ihren Anwendungsfall anpassen müssen, z. B. die AWS Region, mit der eine Verbindung hergestellt wird. Weitere Informationen dazu finden Sie MultiLangDaemon auf GitHub der MultiLangDaemon KCL-Projektseite.

Um die Ruby KCL von herunterzuladen GitHub, gehen Sie zur Kinesis Client Library (Ruby). Den Beispielcode für eine Ruby-KCL-Anwendung für Privatanwender können Sie auf der Projektseite KCL for Ruby unter herunterladen. GitHub

Weitere Informationen über die KCL-Bibliothek für die Unterstützung von Ruby finden Sie unter Dokumentation zu KCL Ruby Gems.

Entwickeln Sie KCL 2.x-Verbraucher

> **Important**
>
> Die Versionen 1.x und 2.x der Amazon Kinesis Client Library (KCL) sind veraltet. KCL 1.x wird am 30. Januar 2026 end-of-support verfügbar sein. Wir empfehlen dringend, dass Sie Ihre KCL-Anwendungen, die Version 1.x verwenden, vor dem 30. Januar 2026 auf die neueste KCL-Version migrieren. Die neueste KCL-Version finden Sie auf der Seite Amazon Kinesis Client Library unter. GitHub Informationen zu den neuesten KCL-Versionen finden Sie unter. Verwenden Sie die Kinesis-Clientbibliothek Informationen zur Migration von KCL 1.x zu KCL 3.x finden Sie unter. Migrieren von KCL 1.x zu KCL 3.x

In diesem Thema erhalten Sie Informationen zur Nutzung der Version 2.0 der Kinesis Client Library (KCL).

Weitere Informationen über die KCL finden Sie in der Übersicht in Entwickeln von Verbrauchern mit der Kinesis Client Library 1.x.

Wählen Sie je nach der Option, die Sie verwenden möchten, aus den folgenden Themen.

Topics

- Entwickeln Sie einen Kinesis Client Library-Consumer in Java
- Entwickeln Sie einen Kinesis Client Library-Consumer in Python
- Entwickeln Sie mit KCL 2.x erweiterte Fan-Out-Nutzer

Entwickeln Sie einen Kinesis Client Library-Consumer in Java

> **Important**
>
> Die Versionen 1.x und 2.x der Amazon Kinesis Client Library (KCL) sind veraltet. KCL 1.x wird end-of-support am 30. Januar 2026 verfügbar sein. Wir empfehlen dringend, dass Sie Ihre KCL-Anwendungen, die Version 1.x verwenden, vor dem 30. Januar 2026 auf die neueste KCL-Version migrieren. Die neueste KCL-Version finden Sie auf der Seite Amazon Kinesis Client Library unter. GitHub Informationen zu den neuesten KCL-Versionen finden Sie unter. Verwenden Sie die Kinesis-Clientbibliothek Informationen zur Migration von KCL 1.x zu KCL 3.x finden Sie unter. Migrieren von KCL 1.x zu KCL 3.x

Der folgende Code zeigt eine Beispielimplementierung in Java für `ProcessorFactory` und `RecordProcessor`. Weitere Informationen zur Nutzung der Vorteile der erweiterten Rundsendefunktion finden Sie unter Verwenden von Verbrauchern mit erweitertem Rundsenden .

```
/*
 *  Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 *  Licensed under the Amazon Software License (the "License").
 *  You may not use this file except in compliance with the License.
 *  A copy of the License is located at
 *
 *  http://aws.amazon.com/asl/
 *
 *  or in the "license" file accompanying this file. This file is distributed
 *  on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 *  express or implied. See the License for the specific language governing
 *  permissions and limitations under the License.
 */


/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
```

```
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 */

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.UUID;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.ScheduledFuture;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

import org.apache.commons.lang3.ObjectUtils;
import org.apache.commons.lang3.RandomStringUtils;
import org.apache.commons.lang3.RandomUtils;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.slf4j.MDC;

import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchAsyncClient;
import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.kinesis.KinesisAsyncClient;
import software.amazon.awssdk.services.kinesis.model.PutRecordRequest;
import software.amazon.kinesis.common.ConfigsBuilder;
import software.amazon.kinesis.common.KinesisClientUtil;
import software.amazon.kinesis.coordinator.Scheduler;
import software.amazon.kinesis.exceptions.InvalidStateException;
import software.amazon.kinesis.exceptions.ShutdownException;
import software.amazon.kinesis.lifecycle.events.InitializationInput;
import software.amazon.kinesis.lifecycle.events.LeaseLostInput;
import software.amazon.kinesis.lifecycle.events.ProcessRecordsInput;
import software.amazon.kinesis.lifecycle.events.ShardEndedInput;
import software.amazon.kinesis.lifecycle.events.ShutdownRequestedInput;
```

```
import software.amazon.kinesis.processor.ShardRecordProcessor;
import software.amazon.kinesis.processor.ShardRecordProcessorFactory;
import software.amazon.kinesis.retrieval.polling.PollingConfig;

/**
 * This class will run a simple app that uses the KCL to read data and uses the AWS SDK
 to publish data.
 * Before running this program you must first create a Kinesis stream through the AWS
 console or AWS SDK.
 */
public class SampleSingle {

    private static final Logger log = LoggerFactory.getLogger(SampleSingle.class);

    /**
     * Invoke the main method with 2 args: the stream name and (optionally) the region.
     * Verifies valid inputs and then starts running the app.
     */
    public static void main(String... args) {
        if (args.length < 1) {
            log.error("At a minimum, the stream name is required as the first argument.
 The Region may be specified as the second argument.");
            System.exit(1);
        }

        String streamName = args[0];
        String region = null;
        if (args.length > 1) {
            region = args[1];
        }

        new SampleSingle(streamName, region).run();
    }

    private final String streamName;
    private final Region region;
    private final KinesisAsyncClient kinesisClient;

    /**
     * Constructor sets streamName and region. It also creates a KinesisClient object
 to send data to Kinesis.
     * This KinesisClient is used to send dummy data so that the consumer has something
 to read; it is also used
     * indirectly by the KCL to handle the consumption of the data.
```

```
     */
    private SampleSingle(String streamName, String region) {
        this.streamName = streamName;
        this.region = Region.of(ObjectUtils.firstNonNull(region, "us-east-2"));
        this.kinesisClient =
KinesisClientUtil.createKinesisAsyncClient(KinesisAsyncClient.builder().region(this.region));
    }

    private void run() {

        /**
         * Sends dummy data to Kinesis. Not relevant to consuming the data with the KCL
         */
        ScheduledExecutorService producerExecutor =
Executors.newSingleThreadScheduledExecutor();
        ScheduledFuture<?> producerFuture =
producerExecutor.scheduleAtFixedRate(this::publishRecord, 10, 1, TimeUnit.SECONDS);

        /**
         * Sets up configuration for the KCL, including DynamoDB and CloudWatch
dependencies. The final argument, a
         * ShardRecordProcessorFactory, is where the logic for record processing lives,
and is located in a private
         * class below.
         */
        DynamoDbAsyncClient dynamoClient =
DynamoDbAsyncClient.builder().region(region).build();
        CloudWatchAsyncClient cloudWatchClient =
CloudWatchAsyncClient.builder().region(region).build();
        ConfigsBuilder configsBuilder = new ConfigsBuilder(streamName, streamName,
kinesisClient, dynamoClient, cloudWatchClient, UUID.randomUUID().toString(), new
SampleRecordProcessorFactory());

        /**
         * The Scheduler (also called Worker in earlier versions of the KCL) is the
entry point to the KCL. This
         * instance is configured with defaults provided by the ConfigsBuilder.
         */
        Scheduler scheduler = new Scheduler(
                configsBuilder.checkpointConfig(),
                configsBuilder.coordinatorConfig(),
                configsBuilder.leaseManagementConfig(),
                configsBuilder.lifecycleConfig(),
                configsBuilder.metricsConfig(),
```

```
                configsBuilder.processorConfig(),
                configsBuilder.retrievalConfig().retrievalSpecificConfig(new
PollingConfig(streamName, kinesisClient))
        );

        /**
         * Kickoff the Scheduler. Record processing of the stream of dummy data will
continue indefinitely
         * until an exit is triggered.
         */
        Thread schedulerThread = new Thread(scheduler);
        schedulerThread.setDaemon(true);
        schedulerThread.start();

        /**
         * Allows termination of app by pressing Enter.
         */
        System.out.println("Press enter to shutdown");
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
        try {
            reader.readLine();
        } catch (IOException ioex) {
            log.error("Caught exception while waiting for confirm. Shutting down.",
ioex);
        }

        /**
         * Stops sending dummy data.
         */
        log.info("Cancelling producer and shutting down executor.");
        producerFuture.cancel(true);
        producerExecutor.shutdownNow();

        /**
         * Stops consuming data. Finishes processing the current batch of data already
received from Kinesis
         * before shutting down.
         */
        Future<Boolean> gracefulShutdownFuture = scheduler.startGracefulShutdown();
        log.info("Waiting up to 20 seconds for shutdown to complete.");
        try {
            gracefulShutdownFuture.get(20, TimeUnit.SECONDS);
        } catch (InterruptedException e) {
            log.info("Interrupted while waiting for graceful shutdown. Continuing.");
```

```
        } catch (ExecutionException e) {
            log.error("Exception while executing graceful shutdown.", e);
        } catch (TimeoutException e) {
            log.error("Timeout while waiting for shutdown.  Scheduler may not have
exited.");
        }
        log.info("Completed, shutting down now.");
    }

    /**
     * Sends a single record of dummy data to Kinesis.
     */
    private void publishRecord() {
        PutRecordRequest request = PutRecordRequest.builder()
                .partitionKey(RandomStringUtils.randomAlphabetic(5, 20))
                .streamName(streamName)
                .data(SdkBytes.fromByteArray(RandomUtils.nextBytes(10)))
                .build();
        try {
            kinesisClient.putRecord(request).get();
        } catch (InterruptedException e) {
            log.info("Interrupted, assuming shutdown.");
        } catch (ExecutionException e) {
            log.error("Exception while sending data to Kinesis. Will try again next
cycle.", e);
        }
    }

    private static class SampleRecordProcessorFactory implements
ShardRecordProcessorFactory {
        public ShardRecordProcessor shardRecordProcessor() {
            return new SampleRecordProcessor();
        }
    }

    /**
     * The implementation of the ShardRecordProcessor interface is where the heart of
the record processing logic lives.
     * In this example all we do to 'process' is log info about the records.
     */
    private static class SampleRecordProcessor implements ShardRecordProcessor {

        private static final String SHARD_ID_MDC_KEY = "ShardId";
```

```
        private static final Logger log =
LoggerFactory.getLogger(SampleRecordProcessor.class);

        private String shardId;

        /**
         * Invoked by the KCL before data records are delivered to the
ShardRecordProcessor instance (via
         * processRecords). In this example we do nothing except some logging.
         *
         * @param initializationInput Provides information related to initialization.
         */
        public void initialize(InitializationInput initializationInput) {
            shardId = initializationInput.shardId();
            MDC.put(SHARD_ID_MDC_KEY, shardId);
            try {
                log.info("Initializing @ Sequence: {}",
initializationInput.extendedSequenceNumber());
            } finally {
                MDC.remove(SHARD_ID_MDC_KEY);
            }
        }

        /**
         * Handles record processing logic. The Amazon Kinesis Client Library will
invoke this method to deliver
         * data records to the application. In this example we simply log our records.
         *
         * @param processRecordsInput Provides the records to be processed as well as
information and capabilities
         *                            related to them (e.g. checkpointing).
         */
        public void processRecords(ProcessRecordsInput processRecordsInput) {
            MDC.put(SHARD_ID_MDC_KEY, shardId);
            try {
                log.info("Processing {} record(s)",
processRecordsInput.records().size());
                processRecordsInput.records().forEach(r -> log.info("Processing record
pk: {} -- Seq: {}", r.partitionKey(), r.sequenceNumber()));
            } catch (Throwable t) {
                log.error("Caught throwable while processing records. Aborting.");
                Runtime.getRuntime().halt(1);
            } finally {
                MDC.remove(SHARD_ID_MDC_KEY);
```

```
            }
        }

        /** Called when the lease tied to this record processor has been lost. Once the
lease has been lost,
         * the record processor can no longer checkpoint.
         *
         * @param leaseLostInput Provides access to functions and data related to the
loss of the lease.
         */
        public void leaseLost(LeaseLostInput leaseLostInput) {
            MDC.put(SHARD_ID_MDC_KEY, shardId);
            try {
                log.info("Lost lease, so terminating.");
            } finally {
                MDC.remove(SHARD_ID_MDC_KEY);
            }
        }

        /**
         * Called when all data on this shard has been processed. Checkpointing must
occur in the method for record
         * processing to be considered complete; an exception will be thrown otherwise.
         *
         * @param shardEndedInput Provides access to a checkpointer method for
completing processing of the shard.
         */
        public void shardEnded(ShardEndedInput shardEndedInput) {
            MDC.put(SHARD_ID_MDC_KEY, shardId);
            try {
                log.info("Reached shard end checkpointing.");
                shardEndedInput.checkpointer().checkpoint();
            } catch (ShutdownException | InvalidStateException e) {
                log.error("Exception while checkpointing at shard end. Giving up.", e);
            } finally {
                MDC.remove(SHARD_ID_MDC_KEY);
            }
        }

        /**
         * Invoked when Scheduler has been requested to shut down (i.e. we decide to
stop running the app by pressing
         * Enter). Checkpoints and logs the data a final time.
         *
```

```
         * @param shutdownRequestedInput Provides access to a checkpointer, allowing a
 record processor to checkpoint
         *                               before the shutdown is completed.
         */
        public void shutdownRequested(ShutdownRequestedInput shutdownRequestedInput) {
            MDC.put(SHARD_ID_MDC_KEY, shardId);
            try {
                log.info("Scheduler is shutting down, checkpointing.");
                shutdownRequestedInput.checkpointer().checkpoint();
            } catch (ShutdownException | InvalidStateException e) {
                log.error("Exception while checkpointing at requested shutdown. Giving
 up.", e);
            } finally {
                MDC.remove(SHARD_ID_MDC_KEY);
            }
        }
    }

}
```

Entwickeln Sie einen Kinesis Client Library-Consumer in Python

> **Important**
>
> Die Versionen 1.x und 2.x der Amazon Kinesis Client Library (KCL) sind veraltet. KCL 1.x wird end-of-support am 30. Januar 2026 verfügbar sein. Wir empfehlen dringend, dass Sie Ihre KCL-Anwendungen, die Version 1.x verwenden, vor dem 30. Januar 2026 auf die neueste KCL-Version migrieren. Die neueste KCL-Version finden Sie auf der Seite Amazon Kinesis Client Library unter. GitHub Informationen zu den neuesten KCL-Versionen finden Sie unter. Verwenden Sie die Kinesis-Clientbibliothek Informationen zur Migration von KCL 1.x zu KCL 3.x finden Sie unter. Migrieren von KCL 1.x zu KCL 3.x

Sie können die Kinesis Client Library (KCL) verwenden, um Anwendungen zu erstellen, die Daten aus Ihren Kinesis-Datenströmen verarbeiten. Die Kinesis Client Library ist in mehreren Sprachen verfügbar. In diesem Thema wird Python behandelt.

Die KCL ist eine Java-Bibliothek. Unterstützung für andere Sprachen als Java wird über eine mehrsprachige Schnittstelle bereitgestellt, die als. MultiLangDaemon Dieser Daemon basiert auf Java und wird im Hintergrund ausgeführt, wenn Sie eine andere KCL-Sprache als Java verwenden. Wenn

Sie also die KCL für Python installieren und Ihre Consumer-App vollständig in Python schreiben, muss Java aufgrund der MultiLangDaemon trotzdem auf Ihrem System installiert sein. Darüber hinaus MultiLangDaemon verfügt es über einige Standardeinstellungen, die Sie möglicherweise an Ihren Anwendungsfall anpassen müssen, z. B. die AWS Region, mit der eine Verbindung hergestellt wird. Weitere Informationen dazu finden Sie MultiLangDaemon auf GitHub der MultiLangDaemon KCL-Projektseite.

Um die Python-KCL von herunterzuladen GitHub, gehen Sie zur Kinesis Client Library (Python). Um Beispielcode für eine Python-KCL-Consumer-Anwendung herunterzuladen, gehen Sie zur KCL for Python-Beispielprojektseite unter. GitHub

Sie müssen die folgenden Aufgaben durchführen, wenn Sie eine KCL-Konsumentenanwendung in Python implementieren:

Aufgaben

- Implementieren Sie die Klassenmethoden RecordProcessor
- Ändern Sie die Konfigurationseigenschaften

Implementieren Sie die Klassenmethoden RecordProcessor

Die `RecordProcess`-Klasse muss die `RecordProcessorBase`-Klasse erweitern, um die folgenden Methoden zu implementieren:

```
initialize
process_records
shutdown_requested
```

Dieses Beispiel stellt Implementierungen bereit, die Sie als Ausgangspunkt verwenden können.

```
#!/usr/bin/env python

# Copyright 2014-2015 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#
# Licensed under the Amazon Software License (the "License").
# You may not use this file except in compliance with the License.
# A copy of the License is located at
#
# http://aws.amazon.com/asl/
#
# or in the "license" file accompanying this file. This file is distributed
```

```
# on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
# express or implied. See the License for the specific language governing
# permissions and limitations under the License.

from __future__ import print_function

import sys
import time

from amazon_kclpy import kcl
from amazon_kclpy.v3 import processor


class RecordProcessor(processor.RecordProcessorBase):
    """
    A RecordProcessor processes data from a shard in a stream. Its methods will be
 called with this pattern:

    * initialize will be called once
    * process_records will be called zero or more times
    * shutdown will be called if this MultiLangDaemon instance loses the lease to this
 shard, or the shard ends due
        a scaling change.
    """
    def __init__(self):
        self._SLEEP_SECONDS = 5
        self._CHECKPOINT_RETRIES = 5
        self._CHECKPOINT_FREQ_SECONDS = 60
        self._largest_seq = (None, None)
        self._largest_sub_seq = None
        self._last_checkpoint_time = None

    def log(self, message):
        sys.stderr.write(message)

    def initialize(self, initialize_input):
        """
        Called once by a KCLProcess before any calls to process_records

        :param amazon_kclpy.messages.InitializeInput initialize_input: Information
 about the lease that this record
            processor has been assigned.
        """
        self._largest_seq = (None, None)
```

```
        self._last_checkpoint_time = time.time()

    def checkpoint(self, checkpointer, sequence_number=None, sub_sequence_number=None):
        """
        Checkpoints with retries on retryable exceptions.

        :param amazon_kclpy.kcl.Checkpointer checkpointer: the checkpointer provided to
either process_records
            or shutdown
        :param str or None sequence_number: the sequence number to checkpoint at.
        :param int or None sub_sequence_number: the sub sequence number to checkpoint
at.
        """
        for n in range(0, self._CHECKPOINT_RETRIES):
            try:
                checkpointer.checkpoint(sequence_number, sub_sequence_number)
                return
            except kcl.CheckpointError as e:
                if 'ShutdownException' == e.value:
                    #
                    # A ShutdownException indicates that this record processor should
be shutdown. This is due to
                    # some failover event, e.g. another MultiLangDaemon has taken the
lease for this shard.
                    #
                    print('Encountered shutdown exception, skipping checkpoint')
                    return
                elif 'ThrottlingException' == e.value:
                    #
                    # A ThrottlingException indicates that one of our dependencies is
is over burdened, e.g. too many
                    # dynamo writes. We will sleep temporarily to let it recover.
                    #
                    if self._CHECKPOINT_RETRIES - 1 == n:
                        sys.stderr.write('Failed to checkpoint after {n} attempts,
giving up.\n'.format(n=n))
                        return
                    else:
                        print('Was throttled while checkpointing, will attempt again in
{s} seconds'
                              .format(s=self._SLEEP_SECONDS))
                elif 'InvalidStateException' == e.value:
                    sys.stderr.write('MultiLangDaemon reported an invalid state while
checkpointing.\n')
```

```
                else:  # Some other error
                    sys.stderr.write('Encountered an error while checkpointing, error
was {e}.\n'.format(e=e))
            time.sleep(self._SLEEP_SECONDS)

    def process_record(self, data, partition_key, sequence_number,
sub_sequence_number):
        """
        Called for each record that is passed to process_records.

        :param str data: The blob of data that was contained in the record.
        :param str partition_key: The key associated with this recod.
        :param int sequence_number: The sequence number associated with this record.
        :param int sub_sequence_number: the sub sequence number associated with this
record.
        """
        ####################################
        # Insert your processing logic here
        ####################################
        self.log("Record (Partition Key: {pk}, Sequence Number: {seq}, Subsequence
Number: {sseq}, Data Size: {ds}"
                 .format(pk=partition_key, seq=sequence_number,
sseq=sub_sequence_number, ds=len(data)))

    def should_update_sequence(self, sequence_number, sub_sequence_number):
        """
        Determines whether a new larger sequence number is available

        :param int sequence_number: the sequence number from the current record
        :param int sub_sequence_number: the sub sequence number from the current record
        :return boolean: true if the largest sequence should be updated, false
otherwise
        """
        return self._largest_seq == (None, None) or sequence_number >
self._largest_seq[0] or \
            (sequence_number == self._largest_seq[0] and sub_sequence_number >
self._largest_seq[1])

    def process_records(self, process_records_input):
        """
        Called by a KCLProcess with a list of records to be processed and a
checkpointer which accepts sequence numbers
        from the records to indicate where in the stream to checkpoint.
```

```
        :param amazon_kclpy.messages.ProcessRecordsInput process_records_input: the
 records, and metadata about the
            records.
        """
        try:
            for record in process_records_input.records:
                data = record.binary_data
                seq = int(record.sequence_number)
                sub_seq = record.sub_sequence_number
                key = record.partition_key
                self.process_record(data, key, seq, sub_seq)
                if self.should_update_sequence(seq, sub_seq):
                    self._largest_seq = (seq, sub_seq)

            #
            # Checkpoints every self._CHECKPOINT_FREQ_SECONDS seconds
            #
            if time.time() - self._last_checkpoint_time >
 self._CHECKPOINT_FREQ_SECONDS:
                self.checkpoint(process_records_input.checkpointer,
 str(self._largest_seq[0]), self._largest_seq[1])
                self._last_checkpoint_time = time.time()

        except Exception as e:
            self.log("Encountered an exception while processing records. Exception was
 {e}\n".format(e=e))

    def lease_lost(self, lease_lost_input):
        self.log("Lease has been lost")

    def shard_ended(self, shard_ended_input):
        self.log("Shard has ended checkpointing")
        shard_ended_input.checkpointer.checkpoint()

    def shutdown_requested(self, shutdown_requested_input):
        self.log("Shutdown has been requested, checkpointing.")
        shutdown_requested_input.checkpointer.checkpoint()


if __name__ == "__main__":
    kcl_process = kcl.KCLProcess(RecordProcessor())
    kcl_process.run()
```

Ändern Sie die Konfigurationseigenschaften

Das Beispiel zeigt Standardwerte für die Konfigurationseigenschaften, wie in dem folgenden Skript gezeigt. Sie können diese Eigenschaften mit eigenen Werten überschreiben.

```
# The script that abides by the multi-language protocol. This script will
# be executed by the MultiLangDaemon, which will communicate with this script
# over STDIN and STDOUT according to the multi-language protocol.
executableName = sample_kclpy_app.py

# The name of an Amazon Kinesis stream to process.
streamName = words

# Used by the KCL as the name of this application. Will be used as the name
# of an Amazon DynamoDB table which will store the lease and checkpoint
# information for workers with this application name
applicationName = PythonKCLSample

# Users can change the credentials provider the KCL will use to retrieve credentials.
# The DefaultAWSCredentialsProviderChain checks several other providers, which is
# described here:
# http://docs.aws.amazon.com/AWSJavaSDK/latest/javadoc/com/amazonaws/auth/
DefaultAWSCredentialsProviderChain.html
AWSCredentialsProvider = DefaultAWSCredentialsProviderChain

# Appended to the user agent of the KCL. Does not impact the functionality of the
# KCL in any other way.
processingLanguage = python/2.7

# Valid options at TRIM_HORIZON or LATEST.
# See http://docs.aws.amazon.com/kinesis/latest/APIReference/
API_GetShardIterator.html#API_GetShardIterator_RequestSyntax
initialPositionInStream = TRIM_HORIZON

# The following properties are also available for configuring the KCL Worker that is
 created
# by the MultiLangDaemon.

# The KCL defaults to us-east-1
#regionName = us-east-1

# Fail over time in milliseconds. A worker which does not renew it's lease within this
 time interval
```

```
# will be regarded as having problems and it's shards will be assigned to other
 workers.
# For applications that have a large number of shards, this msy be set to a higher
 number to reduce
# the number of DynamoDB IOPS required for tracking leases
#failoverTimeMillis = 10000

# A worker id that uniquely identifies this worker among all workers using the same
 applicationName
# If this isn't provided a MultiLangDaemon instance will assign a unique workerId to
 itself.
#workerId =

# Shard sync interval in milliseconds - e.g. wait for this long between shard sync
 tasks.
#shardSyncIntervalMillis = 60000

# Max records to fetch from Kinesis in a single GetRecords call.
#maxRecords = 10000

# Idle time between record reads in milliseconds.
#idleTimeBetweenReadsInMillis = 1000

# Enables applications flush/checkpoint (if they have some data "in progress", but
 don't get new data for while)
#callProcessRecordsEvenForEmptyRecordList = false

# Interval in milliseconds between polling to check for parent shard completion.
# Polling frequently will take up more DynamoDB IOPS (when there are leases for shards
 waiting on
# completion of parent shards).
#parentShardPollIntervalMillis = 10000

# Cleanup leases upon shards completion (don't wait until they expire in Kinesis).
# Keeping leases takes some tracking/resources (e.g. they need to be renewed,
 assigned), so by default we try
# to delete the ones we don't need any longer.
#cleanupLeasesUponShardCompletion = true

# Backoff time in milliseconds for Amazon Kinesis Client Library tasks (in the event of
 failures).
#taskBackoffTimeMillis = 500

# Buffer metrics for at most this long before publishing to CloudWatch.
```

```
#metricsBufferTimeMillis = 10000

# Buffer at most this many metrics before publishing to CloudWatch.
#metricsMaxQueueSize = 10000

# KCL will validate client provided sequence numbers with a call to Amazon Kinesis
 before checkpointing for calls
# to RecordProcessorCheckpointer#checkpoint(String) by default.
#validateSequenceNumberBeforeCheckpointing = true

# The maximum number of active threads for the MultiLangDaemon to permit.
# If a value is provided then a FixedThreadPool is used with the maximum
# active threads set to the provided value. If a non-positive integer or no
# value is provided a CachedThreadPool is used.
#maxActiveThreads = 0
```

Anwendungsname

Die KCL erfordert einen Anwendungsnamen, der unter Ihren Anwendungen sowie den Amazon-DynamoDB-Tabellen in derselben Region eindeutig ist. Sie verwendet den Wert der Anwendungsnamenkonfiguration auf folgende Arten:

- Für mit diesem Anwendungsnamen verknüpfte Auftragnehmer wird angenommen, dass sie gemeinsam im gleichen Stream arbeiten. Diese Worker können auf mehrere Instances verteilt sein. Wenn Sie eine zusätzliche Instance desselben Anwendungscodes ausführen, jedoch mit einem anderen Anwendungsnamen, behandelt die KCL die zweite Instance als eine völlig getrennte Anwendung, die ebenfalls im selben Stream arbeitet.
- Die KCL erstellt eine DynamoDB-Tabelle mit dem Namen der Anwendung und verwendet die Tabelle für die Verwaltung von Statusinformationen für die Anwendung (wie Checkpoints und Auftragnehmer-Shard-Zuweisungen). Jede Anwendung verfügt über eine eigene DynamoDB-Tabelle. Weitere Informationen finden Sie unter Verwenden Sie eine Leasetabelle, um nachzuverfolgen, welche Shards von der KCL-Consumer-Anwendung verarbeitet wurden.

Anmeldeinformationen

Sie müssen Ihre AWS Anmeldeinformationen einem der Anmeldeinformationsanbieter in der Kette der Standardanmeldedienstanbieter zur Verfügung stellen. Sie können die Eigenschaft `AWSCredentialsProvider` verwenden, um einen Anmeldeinformationsanbieter einzurichten. Wenn Sie Ihre Consumer-Anwendung auf einer Amazon EC2 EC2-Instance ausführen, empfehlen wir Ihnen, die Instance mit einer IAM-Rolle zu konfigurieren. AWS Anmeldeinformationen, die die

mit dieser IAM-Rolle verknüpften Berechtigungen widerspiegeln, werden den Anwendungen auf der Instance über deren Instance-Metadaten zur Verfügung gestellt. Dies ist die sicherste Art, Anmeldeinformationen für eine Konsumentenanwendung zu verwalten, die auf einer EC2-Instance ausgeführt wird.

Entwickeln Sie mit KCL 2.x erweiterte Fan-Out-Nutzer

> **Important**
>
> Die Versionen 1.x und 2.x der Amazon Kinesis Client Library (KCL) sind veraltet. KCL 1.x wird am 30. Januar 2026 end-of-support verfügbar sein. Wir empfehlen dringend, dass Sie Ihre KCL-Anwendungen, die Version 1.x verwenden, vor dem 30. Januar 2026 auf die neueste KCL-Version migrieren. Die neueste KCL-Version finden Sie auf der Seite Amazon Kinesis Client Library unter. GitHub Informationen zu den neuesten KCL-Versionen finden Sie unter. Verwenden Sie die Kinesis-Clientbibliothek Informationen zur Migration von KCL 1.x zu KCL 3.x finden Sie unter. Migrieren von KCL 1.x zu KCL 3.x

Verbraucher, die ein erweitertes Rundsenden in Amazon Kinesis Data Streams verwenden, können Datensätze aus einem Datenstrom mit einem dedizierten Durchsatz von bis zu 2 MB Daten pro Sekunde pro Shard empfangen. Diese Art Verbraucher muss nicht mit anderen Verbrauchern konkurrieren, die Daten aus dem Stream empfangen. Weitere Informationen finden Sie unter Entwickeln Sie verbesserte Fan-Out-Verbraucher mit dediziertem Durchsatz.

Sie können die Version 2.0 oder höher der Kinesis Client Library (KCL) verwenden, um Anwendungen zu entwickeln, die erweitertes Rundsenden verwenden, um Daten aus Streams zu empfangen. Die KCL abonniert Ihre Anwendung automatisch für alle Shards eines Streams und stellt sicher, dass Ihre Privatanwenderanwendung mit einem Durchsatzwert von 2 pro Shard lesen kann. MB/sec Weitere Informationen zur Verwendung der KCL ohne Aktivierung von erweitertem Rundsenden finden Sie unter Entwickeln von Verbrauchern mit der Kinesis Client Library 2.0.

Themen

- Entwickeln Sie erweiterte Fan-Out-Consumer mithilfe von KCL 2.x in Java

Entwickeln Sie erweiterte Fan-Out-Consumer mithilfe von KCL 2.x in Java

> **Important**
>
> Die Versionen 1.x und 2.x der Amazon Kinesis Client Library (KCL) sind veraltet. KCL 1.x wird am 30. Januar 2026 end-of-support verfügbar sein. Wir empfehlen dringend, dass Sie Ihre KCL-Anwendungen, die Version 1.x verwenden, vor dem 30. Januar 2026 auf die neueste KCL-Version migrieren. Die neueste KCL-Version finden Sie auf der Seite Amazon Kinesis Client Library unter. GitHub Informationen zu den neuesten KCL-Versionen finden Sie unter. Verwenden Sie die Kinesis-Clientbibliothek Informationen zur Migration von KCL 1.x zu KCL 3.x finden Sie unter. Migrieren von KCL 1.x zu KCL 3.x

Sie können die Version 2.0 oder höher der Kinesis Client Library (KCL) verwenden, um Anwendungen in Amazon Kinesis Data Streams zu entwickeln, die Daten aus Streams mit erweitertem Rundsenden empfangen. Der folgende Code zeigt eine Beispielimplementierung in Java für `ProcessorFactory` und `RecordProcessor`.

Es wird empfohlen, dass Sie `KinesisClientUtil` zum Erstellen von `KinesisAsyncClient` und zum Konfigurieren von `maxConcurrency` in `KinesisAsyncClient` verwenden.

> **Important**
>
> Für den Amazon Kinesis Client kann sich die Latenz möglicherweise signifikant erhöhen, sofern Sie `KinesisAsyncClient` nicht für einen `maxConcurrency`-Wert konfigurieren, der hoch genug ist, um alle Leases plus zusätzliche Verwendungen von `KinesisAsyncClient` zu ermöglichen.

```
/*
 *  Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 *  Licensed under the Amazon Software License (the "License").
 *  You may not use this file except in compliance with the License.
 *  A copy of the License is located at
 *
 *  http://aws.amazon.com/asl/
 *
 *  or in the "license" file accompanying this file. This file is distributed
```

```
 *  on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 *  express or implied. See the License for the specific language governing
 *  permissions and limitations under the License.
 */

/*
 * Copyright 2019 Amazon.com, Inc. or its affiliates. All Rights Reserved.
 *
 * Licensed under the Apache License, Version 2.0 (the "License").
 * You may not use this file except in compliance with the License.
 * A copy of the License is located at
 *
 *     http://www.apache.org/licenses/LICENSE-2.0
 *
 * or in the "license" file accompanying this file. This file is distributed
 * on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
 * express or implied. See the License for the specific language governing
 * permissions and limitations under the License.
 */

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.UUID;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.ScheduledFuture;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.TimeoutException;

import org.apache.commons.lang3.ObjectUtils;
import org.apache.commons.lang3.RandomStringUtils;
import org.apache.commons.lang3.RandomUtils;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.slf4j.MDC;

import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cloudwatch.CloudWatchAsyncClient;
import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.kinesis.KinesisAsyncClient;
```

```
import software.amazon.awssdk.services.kinesis.model.PutRecordRequest;
import software.amazon.kinesis.common.ConfigsBuilder;
import software.amazon.kinesis.common.KinesisClientUtil;
import software.amazon.kinesis.coordinator.Scheduler;
import software.amazon.kinesis.exceptions.InvalidStateException;
import software.amazon.kinesis.exceptions.ShutdownException;
import software.amazon.kinesis.lifecycle.events.InitializationInput;
import software.amazon.kinesis.lifecycle.events.LeaseLostInput;
import software.amazon.kinesis.lifecycle.events.ProcessRecordsInput;
import software.amazon.kinesis.lifecycle.events.ShardEndedInput;
import software.amazon.kinesis.lifecycle.events.ShutdownRequestedInput;
import software.amazon.kinesis.processor.ShardRecordProcessor;
import software.amazon.kinesis.processor.ShardRecordProcessorFactory;

public class SampleSingle {

    private static final Logger log = LoggerFactory.getLogger(SampleSingle.class);

    public static void main(String... args) {
        if (args.length < 1) {
            log.error("At a minimum, the stream name is required as the first argument.
 The Region may be specified as the second argument.");
            System.exit(1);
        }

        String streamName = args[0];
        String region = null;
        if (args.length > 1) {
            region = args[1];
        }

        new SampleSingle(streamName, region).run();
    }

    private final String streamName;
    private final Region region;
    private final KinesisAsyncClient kinesisClient;

    private SampleSingle(String streamName, String region) {
        this.streamName = streamName;
        this.region = Region.of(ObjectUtils.firstNonNull(region, "us-east-2"));
        this.kinesisClient =
 KinesisClientUtil.createKinesisAsyncClient(KinesisAsyncClient.builder().region(this.region));
    }
```

```
   private void run() {
       ScheduledExecutorService producerExecutor =
Executors.newSingleThreadScheduledExecutor();
       ScheduledFuture<?> producerFuture =
producerExecutor.scheduleAtFixedRate(this::publishRecord, 10, 1, TimeUnit.SECONDS);

       DynamoDbAsyncClient dynamoClient =
DynamoDbAsyncClient.builder().region(region).build();
       CloudWatchAsyncClient cloudWatchClient =
CloudWatchAsyncClient.builder().region(region).build();
       ConfigsBuilder configsBuilder = new ConfigsBuilder(streamName, streamName,
kinesisClient, dynamoClient, cloudWatchClient, UUID.randomUUID().toString(), new
SampleRecordProcessorFactory());

       Scheduler scheduler = new Scheduler(
               configsBuilder.checkpointConfig(),
               configsBuilder.coordinatorConfig(),
               configsBuilder.leaseManagementConfig(),
               configsBuilder.lifecycleConfig(),
               configsBuilder.metricsConfig(),
               configsBuilder.processorConfig(),
               configsBuilder.retrievalConfig()
       );

       Thread schedulerThread = new Thread(scheduler);
       schedulerThread.setDaemon(true);
       schedulerThread.start();

       System.out.println("Press enter to shutdown");
       BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
       try {
           reader.readLine();
       } catch (IOException ioex) {
           log.error("Caught exception while waiting for confirm. Shutting down.",
ioex);
       }

       log.info("Cancelling producer, and shutting down executor.");
       producerFuture.cancel(true);
       producerExecutor.shutdownNow();

       Future<Boolean> gracefulShutdownFuture = scheduler.startGracefulShutdown();
       log.info("Waiting up to 20 seconds for shutdown to complete.");
```

```
        try {
            gracefulShutdownFuture.get(20, TimeUnit.SECONDS);
        } catch (InterruptedException e) {
            log.info("Interrupted while waiting for graceful shutdown. Continuing.");
        } catch (ExecutionException e) {
            log.error("Exception while executing graceful shutdown.", e);
        } catch (TimeoutException e) {
            log.error("Timeout while waiting for shutdown. Scheduler may not have
exited.");
        }
        log.info("Completed, shutting down now.");
    }

    private void publishRecord() {
        PutRecordRequest request = PutRecordRequest.builder()
                .partitionKey(RandomStringUtils.randomAlphabetic(5, 20))
                .streamName(streamName)
                .data(SdkBytes.fromByteArray(RandomUtils.nextBytes(10)))
                .build();
        try {
            kinesisClient.putRecord(request).get();
        } catch (InterruptedException e) {
            log.info("Interrupted, assuming shutdown.");
        } catch (ExecutionException e) {
            log.error("Exception while sending data to Kinesis. Will try again next
cycle.", e);
        }
    }

    private static class SampleRecordProcessorFactory implements
ShardRecordProcessorFactory {
        public ShardRecordProcessor shardRecordProcessor() {
            return new SampleRecordProcessor();
        }
    }


    private static class SampleRecordProcessor implements ShardRecordProcessor {

        private static final String SHARD_ID_MDC_KEY = "ShardId";

        private static final Logger log =
LoggerFactory.getLogger(SampleRecordProcessor.class);
```

```
        private String shardId;

        public void initialize(InitializationInput initializationInput) {
            shardId = initializationInput.shardId();
            MDC.put(SHARD_ID_MDC_KEY, shardId);
            try {
                log.info("Initializing @ Sequence: {}",
initializationInput.extendedSequenceNumber());
            } finally {
                MDC.remove(SHARD_ID_MDC_KEY);
            }
        }

        public void processRecords(ProcessRecordsInput processRecordsInput) {
            MDC.put(SHARD_ID_MDC_KEY, shardId);
            try {
                log.info("Processing {} record(s)",
processRecordsInput.records().size());
                processRecordsInput.records().forEach(r -> log.info("Processing record
pk: {} -- Seq: {}", r.partitionKey(), r.sequenceNumber()));
            } catch (Throwable t) {
                log.error("Caught throwable while processing records. Aborting.");
                Runtime.getRuntime().halt(1);
            } finally {
                MDC.remove(SHARD_ID_MDC_KEY);
            }
        }

        public void leaseLost(LeaseLostInput leaseLostInput) {
            MDC.put(SHARD_ID_MDC_KEY, shardId);
            try {
                log.info("Lost lease, so terminating.");
            } finally {
                MDC.remove(SHARD_ID_MDC_KEY);
            }
        }

        public void shardEnded(ShardEndedInput shardEndedInput) {
            MDC.put(SHARD_ID_MDC_KEY, shardId);
            try {
                log.info("Reached shard end checkpointing.");
                shardEndedInput.checkpointer().checkpoint();
            } catch (ShutdownException | InvalidStateException e) {
                log.error("Exception while checkpointing at shard end. Giving up.", e);
```

```
            } finally {
                MDC.remove(SHARD_ID_MDC_KEY);
            }
        }

        public void shutdownRequested(ShutdownRequestedInput shutdownRequestedInput) {
            MDC.put(SHARD_ID_MDC_KEY, shardId);
            try {
                log.info("Scheduler is shutting down, checkpointing.");
                shutdownRequestedInput.checkpointer().checkpoint();
            } catch (ShutdownException | InvalidStateException e) {
                log.error("Exception while checkpointing at requested shutdown. Giving
 up.", e);
            } finally {
                MDC.remove(SHARD_ID_MDC_KEY);
            }
        }
    }

}
```

Migrieren Sie Verbraucher von KCL 1.x zu KCL 2.x

> **Important**
>
> Die Versionen 1.x und 2.x der Amazon Kinesis Client Library (KCL) sind veraltet. KCL 1.x wird am 30. Januar 2026 verfügbar sein. end-of-support Wir empfehlen dringend, dass Sie Ihre KCL-Anwendungen, die Version 1.x verwenden, vor dem 30. Januar 2026 auf die neueste KCL-Version migrieren. Die neueste KCL-Version finden Sie auf der Seite Amazon Kinesis Client Library unter. GitHub Informationen zu den neuesten KCL-Versionen finden Sie unter. Verwenden Sie die Kinesis-Clientbibliothek Informationen zur Migration von KCL 1.x zu KCL 3.x finden Sie unter. Migrieren von KCL 1.x zu KCL 3.x

In diesem Thema werden die Unterschiede zwischen den Versionen 1.x und 2.x der Kinesis Client Library (KCL) erläutert. Außerdem erfahren Sie, wie Sie Ihren Konsumenten von Version 1.x nach Version 2.x der KCL migrieren. Nach der Migration des Clients werden Datensätze vom letzten Checkpoint-Speicherort verarbeitet.

Version 2.0 der KCL enthält die folgenden Schnittstellenänderungen:

KCL-Schnittstellenänderungen

KCL 1.x-Schnittstelle	KCL 2.0-Schnittstelle
`com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor`	`software.amazon.kinesis.processor.ShardRecordProcessor`
`com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFactory`	`software.amazon.kinesis.processor.ShardRecordProcessorFactory`
`com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IShutdownNotificationAware`	Umgewandelt in `software.amazon.kinesis.processor.ShardRecordProcessor`

Themen

- Migrieren Sie den Aufzeichnungsprozessor
- Migrieren Sie den Plattenprozessor ab Werk
- Migrieren Sie den Worker
- Den Amazon Kinesis-Client konfigurieren
- Entfernung von Leerlaufzeiten
- Entfernung der Client-Konfiguration

Migrieren Sie den Aufzeichnungsprozessor

Das folgende Beispiel zeigt einen Datensatzprozessor, der für KCL 1.x implementiert wurde:

```
package com.amazonaws.kcl;

import com.amazonaws.services.kinesis.clientlibrary.exceptions.InvalidStateException;
import com.amazonaws.services.kinesis.clientlibrary.exceptions.ShutdownException;
import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.IRecordProcessorCheckpointer;
import com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IShutdownNotificationAware;
import com.amazonaws.services.kinesis.clientlibrary.lib.worker.ShutdownReason;
```

```
import com.amazonaws.services.kinesis.clientlibrary.types.InitializationInput;
import com.amazonaws.services.kinesis.clientlibrary.types.ProcessRecordsInput;
import com.amazonaws.services.kinesis.clientlibrary.types.ShutdownInput;

public class TestRecordProcessor implements IRecordProcessor,
 IShutdownNotificationAware {
    @Override
    public void initialize(InitializationInput initializationInput) {
        //
        // Setup record processor
        //
    }

    @Override
    public void processRecords(ProcessRecordsInput processRecordsInput) {
        //
        // Process records, and possibly checkpoint
        //
    }

    @Override
    public void shutdown(ShutdownInput shutdownInput) {
        if (shutdownInput.getShutdownReason() == ShutdownReason.TERMINATE) {
            try {
                shutdownInput.getCheckpointer().checkpoint();
            } catch (ShutdownException | InvalidStateException e) {
                throw new RuntimeException(e);
            }
        }
    }

    @Override
    public void shutdownRequested(IRecordProcessorCheckpointer checkpointer) {
        try {
            checkpointer.checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            //
            // Swallow exception
            //
            e.printStackTrace();
        }
    }
}
```

So migrieren Sie die Datensatzprozessorklasse

1. Ändern Sie die Schnittstellen von `com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor` und `com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IShutdownNotificat` folgendermaßen nach `software.amazon.kinesis.processor.ShardRecordProcessor`:

```
// import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
// import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IShutdownNotificationAware;
import software.amazon.kinesis.processor.ShardRecordProcessor;

// public class TestRecordProcessor implements IRecordProcessor,
 IShutdownNotificationAware {
public class TestRecordProcessor implements ShardRecordProcessor {
```

2. Aktualisieren Sie die `import`-Anweisungen für die Methoden `initialize` und `processRecords`.

```
// import com.amazonaws.services.kinesis.clientlibrary.types.InitializationInput;
import software.amazon.kinesis.lifecycle.events.InitializationInput;

//import com.amazonaws.services.kinesis.clientlibrary.types.ProcessRecordsInput;
import software.amazon.kinesis.lifecycle.events.ProcessRecordsInput;
```

3. Ersetzen Sie die Methode `shutdown` durch die folgenden neuen Methoden: `leaseLost`, `shardEnded` und `shutdownRequested`.

```
//    @Override
//    public void shutdownRequested(IRecordProcessorCheckpointer checkpointer) {
//        //
//        // This is moved to shardEnded(...)
//        //
//        try {
//            checkpointer.checkpoint();
//        } catch (ShutdownException | InvalidStateException e) {
//            //
//            // Swallow exception
//            //
//            e.printStackTrace();
```

```
//        }
//    }

    @Override
    public void leaseLost(LeaseLostInput leaseLostInput) {

    }

    @Override
    public void shardEnded(ShardEndedInput shardEndedInput) {
        try {
            shardEndedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            //
            // Swallow the exception
            //
            e.printStackTrace();
        }
    }

//    @Override
//    public void shutdownRequested(IRecordProcessorCheckpointer checkpointer) {
//        //
//        // This is moved to shutdownRequested(ShutdownReauestedInput)
//        //
//        try {
//            checkpointer.checkpoint();
//        } catch (ShutdownException | InvalidStateException e) {
//            //
//            // Swallow exception
//            //
//            e.printStackTrace();
//        }
//    }

    @Override
    public void shutdownRequested(ShutdownRequestedInput shutdownRequestedInput) {
        try {
            shutdownRequestedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            //
            // Swallow the exception
            //
            e.printStackTrace();
```

```
        }
    }
```

Nachstehend finden Sie die aktualisierte Version der Datensatzprozessorklasse.

```
package com.amazonaws.kcl;

import software.amazon.kinesis.exceptions.InvalidStateException;
import software.amazon.kinesis.exceptions.ShutdownException;
import software.amazon.kinesis.lifecycle.events.InitializationInput;
import software.amazon.kinesis.lifecycle.events.LeaseLostInput;
import software.amazon.kinesis.lifecycle.events.ProcessRecordsInput;
import software.amazon.kinesis.lifecycle.events.ShardEndedInput;
import software.amazon.kinesis.lifecycle.events.ShutdownRequestedInput;
import software.amazon.kinesis.processor.ShardRecordProcessor;

public class TestRecordProcessor implements ShardRecordProcessor {
    @Override
    public void initialize(InitializationInput initializationInput) {

    }

    @Override
    public void processRecords(ProcessRecordsInput processRecordsInput) {

    }

    @Override
    public void leaseLost(LeaseLostInput leaseLostInput) {

    }

    @Override
    public void shardEnded(ShardEndedInput shardEndedInput) {
        try {
            shardEndedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            //
            // Swallow the exception
            //
            e.printStackTrace();
        }
```

```
    }

    @Override
    public void shutdownRequested(ShutdownRequestedInput shutdownRequestedInput) {
        try {
            shutdownRequestedInput.checkpointer().checkpoint();
        } catch (ShutdownException | InvalidStateException e) {
            //
            // Swallow the exception
            //
            e.printStackTrace();
        }
    }
}
```

Migrieren Sie den Plattenprozessor ab Werk

Die Datensatzprozessor-Factory ist für das Erstellen von Prozessoren verantwortlich, wenn eine Lease erworben wird. Nachfolgend sehen Sie ein Beispiel für eine KCL-1.x-Factory.

```
package com.amazonaws.kcl;

import com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFactory;

public class TestRecordProcessorFactory implements IRecordProcessorFactory {
    @Override
    public IRecordProcessor createProcessor() {
        return new TestRecordProcessor();
    }
}
```

So migrieren Sie die Datensatzprozessor-Factory

1. Ändern Sie die implementierte Schnittstelle von `com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFa` folgendermaßen nach `software.amazon.kinesis.processor.ShardRecordProcessorFactory`:

   ```
   // import
    com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessor;
   ```

```
import software.amazon.kinesis.processor.ShardRecordProcessor;

// import
 com.amazonaws.services.kinesis.clientlibrary.interfaces.v2.IRecordProcessorFactory;
import software.amazon.kinesis.processor.ShardRecordProcessorFactory;

// public class TestRecordProcessorFactory implements IRecordProcessorFactory {
public class TestRecordProcessorFactory implements ShardRecordProcessorFactory {
```

2. Ändern Sie die Rückgabesignatur für `createProcessor`.

```
// public IRecordProcessor createProcessor() {
public ShardRecordProcessor shardRecordProcessor() {
```

Es folgt ein Beispiel für die Verwendung der Datensatzprozessor-Factory in 2.0:

```
package com.amazonaws.kcl;

import software.amazon.kinesis.processor.ShardRecordProcessor;
import software.amazon.kinesis.processor.ShardRecordProcessorFactory;

public class TestRecordProcessorFactory implements ShardRecordProcessorFactory {
    @Override
    public ShardRecordProcessor shardRecordProcessor() {
        return new TestRecordProcessor();
    }
}
```

Migrieren Sie den Worker

In Version 2.0 des KCL, einer neuen Klasse mit der Bezeichnung `Scheduler`, wird die `Worker`-Klasse ersetzt. Nachfolgend sehen Sie ein Beispiel für einen KCL-1.x-Auftragnehmer.

```
final KinesisClientLibConfiguration config = new KinesisClientLibConfiguration(...)
final IRecordProcessorFactory recordProcessorFactory = new RecordProcessorFactory();
final Worker worker = new Worker.Builder()
    .recordProcessorFactory(recordProcessorFactory)
    .config(config)
    .build();
```

So migrieren Sie den Worker

1. Ändern Sie die `import`-Anweisung für die `Worker`-Klasse in die Import-Anweisungen für die Klassen `Scheduler` und `ConfigsBuilder`.

```
// import com.amazonaws.services.kinesis.clientlibrary.lib.worker.Worker;
import software.amazon.kinesis.coordinator.Scheduler;
import software.amazon.kinesis.common.ConfigsBuilder;
```

2. Erstellen Sie `ConfigsBuilder` und `Scheduler` wie im folgenden Beispiel gezeigt.

   Es wird empfohlen, dass Sie `KinesisClientUtil` zum Erstellen von `KinesisAsyncClient` und zum Konfigurieren von `maxConcurrency` in `KinesisAsyncClient` verwenden.

> **Important**
>
> Für den Amazon Kinesis Client kann sich die Latenz möglicherweise signifikant erhöhen, sofern Sie `KinesisAsyncClient` nicht für einen `maxConcurrency`-Wert konfigurieren, der hoch genug ist, um alle Leases plus zusätzliche Verwendungen von `KinesisAsyncClient` zu ermöglichen.

```
import java.util.UUID;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.cloudwatch.CloudWatchAsyncClient;
import software.amazon.awssdk.services.kinesis.KinesisAsyncClient;
import software.amazon.kinesis.common.ConfigsBuilder;
import software.amazon.kinesis.common.KinesisClientUtil;
import software.amazon.kinesis.coordinator.Scheduler;

...

Region region = Region.AP_NORTHEAST_2;
KinesisAsyncClient kinesisClient =
 KinesisClientUtil.createKinesisAsyncClient(KinesisAsyncClient.builder().region(region));
DynamoDbAsyncClient dynamoClient =
 DynamoDbAsyncClient.builder().region(region).build();
```

```
CloudWatchAsyncClient cloudWatchClient =
 CloudWatchAsyncClient.builder().region(region).build();

ConfigsBuilder configsBuilder = new ConfigsBuilder(streamName, applicationName,
 kinesisClient, dynamoClient, cloudWatchClient, UUID.randomUUID().toString(), new
 SampleRecordProcessorFactory());

Scheduler scheduler = new Scheduler(
    configsBuilder.checkpointConfig(),
    configsBuilder.coordinatorConfig(),
    configsBuilder.leaseManagementConfig(),
    configsBuilder.lifecycleConfig(),
    configsBuilder.metricsConfig(),
    configsBuilder.processorConfig(),
    configsBuilder.retrievalConfig()
    );
```

Den Amazon Kinesis-Client konfigurieren

Mit Version 2.0 der Kinesis Client Library wurde die Konfiguration des Clients aus einer einzelnen Konfigurationsklasse (`KinesisClientLibConfiguration`) in sechs Konfigurationsklassen verlagert. Die folgende Tabelle beschreibt die Migration.

Konfigurationsfelder und ihre neue Klassen

Originalfeld	Neue Konfigurationsklasse	Description
`applicationName`	`ConfigsBuilder`	Der Name für die KCL-Anwendung. Wird als Standard für `tableName` und `consumerName` verwendet.
`tableName`	`ConfigsBuilder`	Ermöglicht das Überschreiben des für die Lease-Tabelle von Amazon DynamoDB verwendeten Tabellennamens.
`streamName`	`ConfigsBuilder`	Der Name des Streams, dessen Datensätze diese Anwendung verarbeitet.
`kinesisEndpoint`	`ConfigsBuilder`	Diese Option wurde entfernt. Siehe "Entfernen von Client-Konfigurationen".

Originalfeld	Neue Konfigurationsklasse	Description
`dynamoDBEndpoint`	`ConfigsBuilder`	Diese Option wurde entfernt. Siehe "Entfernen von Client-Konfigurationen".
`initialPositionInStreamExtended`	`RetrievalConfig`	Der Speicherort im Shard, an dem die KCL mit der Ausführung der Anwendung durch Abrufen von Datensätzen beginnt.
`kinesisCredentialsProvider`	`ConfigsBuilder`	Diese Option wurde entfernt. Siehe "Entfernen von Client-Konfigurationen".
`dynamoDBCredentialsProvider`	`ConfigsBuilder`	Diese Option wurde entfernt. Siehe "Entfernen von Client-Konfigurationen".
`cloudWatchCredentialsProvider`	`ConfigsBuilder`	Diese Option wurde entfernt. Siehe "Entfernen von Client-Konfigurationen".
`failoverTimeMillis`	LeaseManagementConfig	Anzahl Millisekunden, nach deren Ablauf unterstellt werden kann, dass ein Lease-Eigentümer fehlgeschlagen ist.
`workerIdentifier`	`ConfigsBuilder`	Eine eindeutige Kennung, die die Instanziierung des Anwendungsprozessors repräsentiert. Dieser Wert muss eindeutig sein.
`shardSyncIntervalMillis`	LeaseManagementConfig	Die Zeit zwischen Shard-Synchronisierungsaufrufen.
`maxRecords`	`PollingConfig`	Ermöglicht das Einstellen der maximalen Anzahl an Datensätzen, die Kinesis zurückgibt.

Originalfeld	Neue Konfigurationsklasse	Description
`idleTimeBetweenReadsInMillis`	`CoordinatorConfig`	Diese Option wurde entfernt. Siehe "Entfernen der Leerlaufzeit".
`callProcessRecordsEvenForEmptyRecordList`	`ProcessorConfig`	Wenn diese Option aktiviert ist, wird der Datensatzprozessor aufgerufen, auch wenn Kinesis keine Datensätze bereitgestellt hat.
`parentShardPollIntervalMillis`	`CoordinatorConfig`	Gibt an, wie oft ein Datensatzprozessor abfragen soll, ob der übergeordnete Shard abgeschlossen wurde.
`cleanupLeasesUponShardCompletion`	`LeaseManagementConfig`	Wenn diese Option aktiviert ist, werden Leases entfernt, sobald die untergeordneten Leases die Verarbeitung gestartet haben.
`ignoreUnexpectedChildShards`	`LeaseManagementConfig`	Wenn diese Option aktiviert ist, werden untergeordnete Shards, die einen offenen Shard aufweisen, ignoriert. Dies gilt hauptsächlich für DynamoDB Streams.
`kinesisClientConfig`	`ConfigsBuilder`	Diese Option wurde entfernt. Siehe "Entfernen von Client-Konfigurationen".
`dynamoDBClientConfig`	`ConfigsBuilder`	Diese Option wurde entfernt. Siehe "Entfernen von Client-Konfigurationen".
`cloudWatchClientConfig`	`ConfigsBuilder`	Diese Option wurde entfernt. Siehe "Entfernen von Client-Konfigurationen".
`taskBackoffTimeMillis`	`LifecycleConfig`	Die Wartezeit beim erneuten Versuchen von fehlgeschlagenen Aufgaben.

Originalfeld	Neue Konfigurationsklasse	Description
`metricsBufferTimeMillis`	`MetricsConfig`	Steuert die Veröffentlichung von CloudWatch Metriken.
`metricsMaxQueueSize`	`MetricsConfig`	Steuert die Veröffentlichung von CloudWatch Metriken.
`metricsLevel`	`MetricsConfig`	Steuert die Veröffentlichung von CloudWatch Metriken.
`metricsEnabledDimensions`	`MetricsConfig`	Steuert die Veröffentlichung von CloudWatch Metriken.
`validateSequenceNumberBeforeCheckpointing`	`CheckpointConfig`	Diese Option wurde entfernt. Siehe "Checkpoint Sequence Number Validation".
`regionName`	`ConfigsBuilder`	Diese Option wurde entfernt. Siehe "Entfernen von Client-Konfigurationen".
`maxLeasesForWorker`	`LeaseManagementConfig`	Die maximale Anzahl der Leases, die eine einzelne Instance der Anwendung akzeptieren sollte.
`maxLeasesToStealAtOneTime`	`LeaseManagementConfig`	Die maximale Anzahl der Leases, die eine Anwendung zu einem gegebenen Zeitpunkt zu stehlen versuchen sollte.
`initialLeaseTableReadCapacity`	`LeaseManagementConfig`	Der DynamoDB-Lesevorgang IOPs , der verwendet wird, wenn die Kinesis-Clientbibliothek eine neue DynamoDB-Leasetabelle erstellen muss.

Originalfeld	Neue Konfigurationsklasse	Description
`initialLeaseTableWriteCapacity`	`LeaseManagementConfig`	Der DynamoDB-Lesevorgang IOPs , der verwendet wird, wenn die Kinesis-Clientbibliothek eine neue DynamoDB-Leasetabelle erstellen muss.
`initialPositionInStreamExtended`	LeaseManagementConfig	Die ursprüngliche Position im Stream, an der die Anwendung starten soll. Dieser Wert wird nur im Rahmen der Lease-Erstellung verwendet.
`skipShardSyncAtWorkerInitializationIfLeasesExist`	`CoordinatorConfig`	Synchronisieren der Shard-Daten deaktivieren, wenn die Lease-Tabelle Leases enthält. KinesisEcoTODO: -438
`shardPrioritization`	`CoordinatorConfig`	Welche Shard-Priorisierung verwendet werden soll.
`shutdownGraceMillis`	–	Diese Option wurde entfernt. Siehe Umzüge. MultiLang
`timeoutInSeconds`	–	Diese Option wurde entfernt. Siehe MultiLang Umzüge.
`retryGetRecordsInSeconds`	`PollingConfig`	Konfiguriert die Verzögerung zwischen GetRecords Fehlversuchen.
`maxGetRecordsThreadPool`	`PollingConfig`	Die Thread-Pool-Größe, die für GetRecords verwendet wird.
`maxLeaseRenewalThreads`	`LeaseManagementConfig`	Steuert die Größe des Lease-Renewer-Thread-Pools. Dieser Pool muss größer sein, wenn die Anwendung mehr Leases annehmen kann.

Originalfeld	Neue Konfigurationsklasse	Description
`recordsFetcherFactory`	`PollingConfig`	Ermöglicht es, die Voreinstellung zu ersetzen, die zum Erstellen von Fetchers verwendet wird, die von Streams abrufen.
`logWarningForTaskAfterMillis`	`LifecycleConfig`	Wartezeit, bevor eine Warnung protokolliert wird, wenn eine Aufgabe nicht abgeschlossen wurde.
`listShardsBackoffTimeInMillis`	`RetrievalConfig`	Die Anzahl der zwischen Aufrufen von `ListShards` abzuwartenden Millisekunden, wenn es zu Fehlern kommt.
`maxListShardsRetryAttempts`	`RetrievalConfig`	Die maximale Anzahl Wiederholungsversuche durch `ListShards` , bevor abgebrochen wird.

Entfernung von Leerlaufzeiten

In Version 1.x von KCL wurde `idleTimeBetweenReadsInMillis` für zwei Werte verwendet:

- Zeitraum zwischen Task-Versandprüfungen. Sie können diesen Zeitraum zwischen Tasks jetzt mit `CoordinatorConfig#shardConsumerDispatchPollIntervalMillis` konfigurieren.
- Zeit im Ruhemodus, wenn keine Datensätze von Kinesis Data Streams zurückgegeben wurden. In Version 2.0 werden Datensätze im Rahmen der verbesserten Rundsendung von den jeweiligen Abrufern im Push-Verfahren übermittelt. Aktivitäten auf dem Shard-Konsumenten treten nur auf, wenn eine Anforderung per Push ankommt.

Entfernung der Client-Konfiguration

In Version 2.0 erstellt KCL keine Clients mehr. Der Benutzer muss einen gültigen Client bereitstellen. Mit dieser Änderung wurden alle Konfigurationsparameter für die Client-Erstellung entfernt. Wenn Sie diese Parameter benötigen, können Sie sie in den Clients einstellen, bevor die Clients für `ConfigsBuilder` bereitgestellt werden.

Entferntes Field	Äquivalente Konfiguration
`kinesisEndpoint`	Konfigurieren Sie das SDK `KinesisAsyncClient` mit dem bevorzugten Endpunkt: `KinesisAsyncClient.builder().endpointOverride(URI.create("https://<kinesis endpoint>")).build()` .
`dynamoDBEndpoint`	Konfigurieren Sie das SDK `DynamoDbAsyncClient` mit dem bevorzugten Endpunkt: `DynamoDbAsyncClient.builder().endpointOverride(URI.create("https://<dynamodb endpoint>")).build()` .
`kinesisClientConfig`	Konfigurieren Sie das SDK `KinesisAsyncClient` mit der benötigten Konfiguration: `KinesisAsyncClient.builder().overrideConfiguration(<your configuration>).build()` .
`dynamoDBClientConfig`	Konfigurieren Sie das SDK `DynamoDbAsyncClient` mit der benötigten Konfiguration: `DynamoDbAsyncClient.builder().overrideConfiguration(<your configuration>).build()` .
`cloudWatchClientConfig`	Konfigurieren Sie das SDK `CloudWatchAsyncClient` mit der benötigten Konfiguration: `CloudWatchAsyncClient.builder().overrideConfiguration(<your configuration>).build()` .
`regionName`	Konfigurieren Sie das SDK mit der bevorzugten Region. Dies ist für alle SDK-Clients gleich. Beispiel, `KinesisAsyncClient.builder().region(Region.US_WEST_2).build()` .

Entwickeln Sie Verbraucher mit dem AWS SDK für Java

Mithilfe der Amazon Kinesis Data Streams APIs können Sie benutzerdefinierte Verbraucher entwickeln. In diesem Abschnitt wird die Verwendung der Kinesis Data Streams APIs mit dem AWS SDK für Java beschrieben.

> **Important**
>
> Die empfohlene Methode zur Entwicklung benutzerdefinierter Verbraucher von Kinesis Data Streams mit durchgängiger gemeinsamer Nutzung ist die Verwendung der Kinesis

Client Library (KCL). KCL unterstützt Sie bei der Nutzung und Verarbeitung von Daten aus einem Kinesis-Datenstrom, indem sie sich um viele der komplexen Aufgaben kümmert, die mit verteilter Datenverarbeitung verbunden sind. Weitere Informationen finden Sie unter Entwickeln Sie Verbraucher mit KCL in Java.

Themen

- Entwickeln Sie Verbraucher mit gemeinsamem Durchsatz mit dem AWS SDK für Java
- Entwickeln Sie mehr Fan-Out-Nutzer mit dem AWS SDK für Java
- Interagieren Sie mit Daten mithilfe der Schema Registry AWS Glue

Entwickeln Sie Verbraucher mit gemeinsamem Durchsatz mit dem AWS SDK für Java

Eine der Methoden zur Entwicklung benutzerdefinierter Kinesis Data Streams-Verbraucher, die durchgehend gemeinsam genutzt werden, ist die Verwendung von Amazon Kinesis Data Streams APIs mit dem. AWS SDK für Java In diesem Abschnitt wird die Verwendung der Kinesis Data Streams APIs mit dem AWS SDK für Java beschrieben. Sie können die Kinesis Data Streams APIs mit anderen Programmiersprachen aufrufen. Weitere Informationen zu allen verfügbaren AWS SDKs Produkten finden Sie unter Start Developing with Amazon Web Services.

Der Java-Beispielcode in diesem Abschnitt zeigt, wie grundlegende Kinesis Data Streams Streams-API-Operationen ausgeführt werden. Er ist logisch nach Operationstypen unterteilt. Diese Beispiele stellen keinen produktionsbereiten Code dar. Sie überprüfen nicht alle möglichen Ausnahmen und es werden nicht alle möglichen Sicherheits- oder Leistungsüberlegungen berücksichtigt.

Themen

- Daten aus einem Stream abrufen
- Verwenden Sie Shard-Iteratoren
- Benutze GetRecords
- Passt euch an einen Reshard an

Daten aus einem Stream abrufen

Die Kinesis Data Streams APIs umfassen die `getRecords` Methoden `getShardIterator` und, die Sie aufrufen können, um Datensätze aus einem Datenstream abzurufen. Dies ist das PULL-Modell, bei dem der Code Datensätze direkt aus den Shards des Datenstroms abruft.

> **Important**
>
> Wir empfehlen Ihnen, die von KCL bereitgestellte Unterstützung für Datensatzprozessoren zu verwenden, um Datensätze aus Ihren Datenströmen abzurufen. Dies ist das PUSH-Modell, bei dem Sie den Code implementieren, der die Daten verarbeitet. Die KCL ruft Datensätze aus dem Datenstrom ab und übermittelt diese an Ihren Anwendungscode. Darüber hinaus bietet die KCL Funktionen für einen Failover, eine Wiederherstellung und eine Lastverteilung. Weitere Informationen finden Sie unter Entwickeln benutzerdefinierter Verbraucher mit gemeinsamem Durchsatz mithilfe von KCL.

In einigen Fällen ziehen Sie es jedoch möglicherweise vor, die Kinesis Data Streams APIs zu verwenden. Dies ist beispielsweise der Fall, wenn Sie benutzerdefinierte Tools für das Überwachen und Debuggen Ihrer Datenströme implementieren.

> **Important**
>
> Kinesis Data Streams unterstützt Änderungen des Zeitraums der Datensatzaufbewahrung für einen Datenstrom. Weitere Informationen finden Sie unter Ändern Sie den Aufbewahrungszeitraum für Daten.

Verwenden Sie Shard-Iteratoren

Sie rufen Datensätze aus dem Stream pro Shard ab. Für jeden Shard und jeden Datensatzstapel, den Sie aus dem Shard abrufen, benötigen Sie einen Shard-Iterator. Der Shard-Iterator wird im `getRecordsRequest`-Objekt verwendet, um den Shard anzugeben, aus dem die Datensätze abgerufen werden. Der Typ, der dem Shard-Iterator zugeordnet ist, gibt die Stelle im Shard an, von der die Datensätze abgerufen werden sollen (weitere Informationen dazu finden Sie später in diesem Abschnitt). Bevor Sie mit dem Shard-Iterator arbeiten können, müssen Sie den Shard abrufen. Weitere Informationen finden Sie unter Shards auflisten.

Rufen Sie diesen ersten Shard-Iterator mit der `getShardIterator`-Methode ab. Rufen Sie Shard-Iteratoren für weitere Datensatzstapel mit der `getNextShardIterator` -Methode des `getRecordsResult`-Objekts ab, das von der `getRecords`-Methode zurückgegeben wird. Ein Shard-Iterator verliert seine Gültigkeit nach 5 Minuten. Wenn Sie einen Shard-Iterator verwenden, solange er gültig ist, erhalten Sie einen neuen. Jeder Shard-Iterator ist 5 Minuten lang gültig, selbst wenn er bereits verwendet wurde.

Instanziieren Sie `GetShardIteratorRequest`, um den ersten Shard-Iterator abzurufen. Übergeben Sie ihn an die `getShardIterator`-Methode. Geben Sie zum Konfigurieren der Anforderung den Stream und die Shard-ID an. Informationen darüber, wie Sie die Streams in Ihrem AWS Konto abrufen können, finden Sie unter. Auflisten von Streams Informationen zum Abrufen der Shards in einem Stream finden Sie unter Shards auflisten.

```
String shardIterator;
GetShardIteratorRequest getShardIteratorRequest = new GetShardIteratorRequest();
getShardIteratorRequest.setStreamName(myStreamName);
getShardIteratorRequest.setShardId(shard.getShardId());
getShardIteratorRequest.setShardIteratorType("TRIM_HORIZON");

GetShardIteratorResult getShardIteratorResult =
 client.getShardIterator(getShardIteratorRequest);
shardIterator = getShardIteratorResult.getShardIterator();
```

Der Beispiel-Code gibt beim Abrufen des ersten Shard-Iterators `TRIM_HORIZON` als Iterator-Typ an. Dieser Iterator-Typ bedeutet, dass Datensätze zurückgegeben werden sollen. Und zwar beginnend mit dem ersten zum Shard hinzugefügten Datensatz und nicht mit dem letzten hinzugefügten Datensatz, auch als Spitze bezeichnet. Folgende Iterator-Typen werden unterstützt:

- `AT_SEQUENCE_NUMBER`
- `AFTER_SEQUENCE_NUMBER`
- `AT_TIMESTAMP`
- `TRIM_HORIZON`
- `LATEST`

Weitere Informationen finden Sie unter ShardIteratorType.

Bei einigen Iterator-Typen müssen Sie zusätzlich eine Sequenznummer angeben, z. B.:

```
getShardIteratorRequest.setShardIteratorType("AT_SEQUENCE_NUMBER");
getShardIteratorRequest.setStartingSequenceNumber(specialSequenceNumber);
```

Nachdem Sie einen Datensatz mit `getRecords` abgerufen haben, erhalten Sie die Sequenznummer für den Datensatz, indem Sie die `getSequenceNumber`-Methode des Datensatzes aufrufen.

```
record.getSequenceNumber()
```

Darüber hinaus erhält der Code, durch den Datensätze zum Stream hinzufügt werden, die Sequenznummer für einen hinzugefügten Datensatz, indem `getSequenceNumber` auf dem Ergebnis von `putRecord` aufgerufen wird.

```
lastSequenceNumber = putRecordResult.getSequenceNumber();
```

Sie können mit diesen Sequenznummern eine strenge aufsteigende Anordnung der Datensätze gewährleisten. Weitere Informationen finden Sie im Code-Beispiel unter PutRecord Beispiel.

Benutze GetRecords

Instanziieren Sie nach Abruf des Shard-Iterators ein `GetRecordsRequest`-Objekt. Geben Sie den Iterator für die Anforderung mit der `setShardIterator`-Methode an.

Optional können Sie auch die Anzahl der abzurufenden Datensätze mithilfe der `setLimit`-Methode angeben. Die Anzahl der Datensätze, die `getRecords` zurückgibt, ist stets gleich oder kleiner als dieses Limit. Wenn Sie kein Limit angeben, gibt `getRecords` 10 MB abgerufener Datensätze zurück. Beim unten stehenden Beispiel-Code wird das Limit auf 25 Datensätze festgelegt.

Wenn keine Datensätze zurückgegeben werden, bedeutet dies, dass derzeit keine Datensätze von diesem Shard für die vom Shard-Iterator angegebene Sequenznummer verfügbar sind. Wenn dies der Fall ist, sollte Ihre Anwendung so lange warten, wie dies für die Datenquellen des Streams angemessen ist. Versuchen Sie dann erneut mit dem Shard-Iterator, der vom vorherigen Aufruf von `getRecords` zurückgegeben wurde, Daten aus dem Shard abzurufen.

Übergeben Sie `getRecordsRequest` an die `getRecords`-Methode und erfassen Sie die zurückgegebenen Werte als `getRecordsResult`-Objekt. Rufen Sie die `getRecords`-Methode auf dem `getRecordsResult`-Objekt auf, um die Datensätze abzurufen.

```
GetRecordsRequest getRecordsRequest = new GetRecordsRequest();
getRecordsRequest.setShardIterator(shardIterator);
getRecordsRequest.setLimit(25);

GetRecordsResult getRecordsResult = client.getRecords(getRecordsRequest);
List<Record> records = getRecordsResult.getRecords();
```

Zur Vorbereitung auf einen weiteren Aufruf von `getRecords` rufen Sie den nächsten Shard-Iterator von `getRecordsResult` ab.

```
shardIterator = getRecordsResult.getNextShardIterator();
```

Die besten Ergebnisse erzielen Sie, wenn Sie mindestens 1 Sekunde (1.000 Millisekunden) zwischen den Aufrufen von `getRecords` warten, um ein Überschreiten des Limits für die Aufrufe von `getRecords` zu vermeiden.

```
try {
  Thread.sleep(1000);
}
catch (InterruptedException e) {}
```

In der Regel sollten Sie `getRecords` in einer Schleife aufrufen, auch wenn Sie einen einzelnen Datensatz in einem Testszenario abrufen. Ein einzelner Aufruf von `getRecords` gibt möglicherweise eine leere Datensatzliste zurück, auch wenn der Shard mehrere Datensätze mit höheren Sequenznummern enthält. In diesem Fall verweist der zurückgegebene `NextShardIterator` zusammen mit der leeren Datensatzliste auf eine höhere Sequenznummer im Shard. Nachfolgende Aufrufe von `getRecords` führen dann zu einem erfolgreichen Abruf. Das folgende Beispiel zeigt die Verwendung einer Schleife.

Beispiel: getRecords

Das folgende Codebeispiel spiegelt die `getRecords`-Tipps in diesem Abschnitt wieder, einschließlich der Aufrufe in Schleifen.

```
// Continuously read data records from a shard
List<Record> records;

while (true) {

  // Create a new getRecordsRequest with an existing shardIterator
  // Set the maximum records to return to 25

  GetRecordsRequest getRecordsRequest = new GetRecordsRequest();
  getRecordsRequest.setShardIterator(shardIterator);
  getRecordsRequest.setLimit(25);

  GetRecordsResult result = client.getRecords(getRecordsRequest);

  // Put the result into record list. The result can be empty.
  records = result.getRecords();

  try {
    Thread.sleep(1000);
  }
  catch (InterruptedException exception) {
    throw new RuntimeException(exception);
  }

  shardIterator = result.getNextShardIterator();
}
```

Bei der Nutzung der Kinesis Client Library müssen möglicherweise mehrere Aufrufe durchgeführt werden, ehe Daten zurückgegeben werden. Dieses Verhalten ist Design-bedingt und kein Fehler der KCL oder Ihrer Daten.

Passt euch an einen Reshard an

Wenn `getRecordsResult.getNextShardIterator` `null` zurückgibt, bedeutet dies, dass eine Aufteilung oder Zusammenführung des Shards stattgefunden hat, die diesen Shard betrifft. Dieser Shard befindet sich jetzt in einem `CLOSED`-Status und Sie haben alle verfügbaren Datensätze von diesem Shard gelesen.

In diesem Szenario können Sie `getRecordsResult.childShards` verwenden, um etwas über die neuen untergeordneten Shards des zu verarbeitenden Shards zu erfahren, die durch die Aufteilung oder Zusammenführung entstanden sind. Weitere Informationen finden Sie unter ChildShard.

Bei einer Teilung ist die `parentShardId` der beiden neuen Shards gleich der Shard-ID des zuvor verarbeiteten Shards. Der Wert von `adjacentParentShardId` für beide Shards ist `null`.

Bei einer Zusammenführung ist bei dem entstandenen einzelnen Datensatz die `parentShardId` identisch mit der Shard-ID eines übergeordneten Shards und die `adjacentParentShardId` ist gleich der Shard-ID des anderen übergeordneten Shards. Ihre Anwendung hat bereits alle Daten aus einem dieser Shards ausgelesen. Dies ist der Shard, für den `getRecordsResult.getNextShardIterator` `null` zurückgegeben hat. Wenn die Reihenfolge der Daten für Ihre Anwendung von Bedeutung ist, stellen Sie sicher, dass die Anwendung auch alle Daten des anderen übergeordneten Shards ausliest, ehe neue Daten aus dem durch die Zusammenführung entstandenen untergeordneten Shards ausgelesen werden.

Wenn Sie mehrere Prozessoren zum Abrufen von Daten aus dem Stream verwenden (beispielsweise einen Prozessor pro Shard), und es kommt zu einer Teilung oder Zusammenführung von Shards, sollten Sie die Anzahl der Prozessoren entsprechend anpassen.

Weitere Informationen zum Resharding, einschließlich einer Diskussion über Shard-Status, beispielsweise `CLOSED` finden Sie unter Einen Stream erneut teilen.

Entwickeln Sie mehr Fan-Out-Nutzer mit dem AWS SDK für Java

Erweitertes Rundsenden ist ein Feature in Amazon Kinesis Data Streams, die es Verbrauchern ermöglicht, Datensätze aus einem Datenstrom mit einem dedizierten Durchsatz von bis zu 2 MB Daten pro Sekunde pro Shard zu empfangen. Ein Verbraucher, der ein erweitertes Rundsenden verwendet, muss nicht mit anderen Verbrauchern konkurrieren, die Daten aus dem Stream empfangen. Weitere Informationen finden Sie unter Entwickeln Sie verbesserte Fan-Out-Verbraucher mit dediziertem Durchsatz.

Sie können API-Operationen zum Erstellen eines Verbrauchers in Kinesis Data Streams verwenden der erweitertes Rundsenden verwendet.

Einen Verbraucher mit erweitertem Rundsenden unter Verwendung der API für Kinesis Data Streams registrieren

1. Rufen Sie an RegisterStreamConsumer, um Ihre Anwendung als Endverbraucher zu registrieren, der den erweiterten Fan-Out verwendet. Kinesis Data Streams generiert einen Amazon-Ressourcennamen (ARN) für den Verbraucher und gibt ihn in der Antwort zurück.
2. Um mit dem Abhören eines bestimmten Shards zu beginnen, geben Sie den Kunden-ARN in einem Anruf an SubscribeToShardweiter. Kinesis Data Streams beginnt dann, die Datensätze

von diesem Shard in Form von Ereignissen des Typs SubscribeToShardEventüber eine HTTP/2-Verbindung an Sie weiterzuleiten. Die Verbindung bleibt für bis zu 5 Minuten offen. Rufen Sie SubscribeToSharderneut an, wenn Sie weiterhin Datensätze von dem Shard empfangen möchten`future`, nachdem der vom Anruf zurückgegebene Shard normal oder ausnahmsweise SubscribeToShardabgeschlossen wurde.

> **Note**
>
> Die `SubscribeToShard`-API gibt auch die Liste der untergeordneten Shards des aktuellen Shards zurück, wenn das Ende des aktuellen Shards erreicht ist.

3. Rufen Sie an, um einen Verbraucher abzumelden, der den erweiterten Fan-Out verwendet. DeregisterStreamConsumer

Der folgende Code ist ein Beispiel dafür, wie Sie für Ihren Verbraucher ein Abonnement für einen Shard einrichten, das Abonnement regelmäßig erneuern und die Ereignisse verarbeiten können.

```
    import software.amazon.awssdk.services.kinesis.KinesisAsyncClient;
    import software.amazon.awssdk.services.kinesis.model.ShardIteratorType;
    import software.amazon.awssdk.services.kinesis.model.SubscribeToShardEvent;
    import software.amazon.awssdk.services.kinesis.model.SubscribeToShardRequest;
    import
 software.amazon.awssdk.services.kinesis.model.SubscribeToShardResponseHandler;

    import java.util.concurrent.CompletableFuture;

    /**
     * See https://github.com/awsdocs/aws-doc-sdk-examples/blob/master/javav2/
example_code/kinesis/src/main/java/com/example/kinesis/KinesisStreamEx.java
     * for complete code and more examples.
     */
    public class SubscribeToShardSimpleImpl {

        private static final String CONSUMER_ARN = "arn:aws:kinesis:us-
east-1:123456789123:stream/foobar/consumer/test-consumer:1525898737";
        private static final String SHARD_ID = "shardId-000000000000";

        public static void main(String[] args) {

            KinesisAsyncClient client = KinesisAsyncClient.create();
```

```
            SubscribeToShardRequest request = SubscribeToShardRequest.builder()
                    .consumerARN(CONSUMER_ARN)
                    .shardId(SHARD_ID)
                    .startingPosition(s -> s.type(ShardIteratorType.LATEST)).build();

            // Call SubscribeToShard iteratively to renew the subscription
periodically.
            while(true) {
                // Wait for the CompletableFuture to complete normally or
exceptionally.
                callSubscribeToShardWithVisitor(client, request).join();
            }

            // Close the connection before exiting.
            // client.close();
        }


        /**
         * Subscribes to the stream of events by implementing the
SubscribeToShardResponseHandler.Visitor interface.
         */
        private static CompletableFuture<Void>
callSubscribeToShardWithVisitor(KinesisAsyncClient client, SubscribeToShardRequest
request) {
            SubscribeToShardResponseHandler.Visitor visitor = new
SubscribeToShardResponseHandler.Visitor() {
                @Override
                public void visit(SubscribeToShardEvent event) {
                    System.out.println("Received subscribe to shard event " + event);
                }
            };
            SubscribeToShardResponseHandler responseHandler =
SubscribeToShardResponseHandler
                    .builder()
                    .onError(t -> System.err.println("Error during stream - " +
t.getMessage()))
                    .subscriber(visitor)
                    .build();
            return client.subscribeToShard(request, responseHandler);
        }
    }
```

Wenn `event.ContinuationSequenceNumber` `null` zurückgibt, bedeutet dies, dass eine Aufteilung oder Zusammenführung des Shards stattgefunden hat, die diesen Shard betrifft. Dieser Shard befindet sich jetzt in einem `CLOSED`-Status und Sie haben alle verfügbaren Datensätze von diesem Shard gelesen. In diesem Szenario können Sie, wie im obigen Beispiel, `event.childShards` verwenden, um etwas über die neuen untergeordneten Shards des zu verarbeitenden Shards zu erfahren, die durch die Aufteilung oder Zusammenführung entstanden sind. Weitere Informationen finden Sie unter ChildShard.

Interagieren Sie mit Daten mithilfe der Schema Registry AWS Glue

Sie können Ihre Kinesis-Datenströme in die AWS Glue Schema Registry integrieren. Mit der AWS Glue Schema Registry können Sie Schemas zentral erkennen, steuern und weiterentwickeln und gleichzeitig sicherstellen, dass die erstellten Daten kontinuierlich anhand eines registrierten Schemas validiert werden. Ein Schema definiert die Struktur und das Format eines Datensatzes. Ein Schema ist eine versionierte Spezifikation für zuverlässige Datenveröffentlichung, -nutzung oder -speicherung. Mit der AWS Glue Schema Registry können Sie die end-to-end Datenqualität und Datenverwaltung in Ihren Streaming-Anwendungen verbessern. Weitere Informationen finden Sie unter AWS Glue Schema Registry. Eine Möglichkeit, diese Integration einzurichten, ist die `GetRecords` Kinesis Data Streams Streams-API, die im AWS Java SDK verfügbar ist.

Detaillierte Anweisungen zur Einrichtung der Integration von Kinesis Data Streams mit Schema Registry mithilfe der `GetRecords` Kinesis Data Streams APIs finden Sie im Abschnitt „Interaktion mit Daten mithilfe der Kinesis Data Streams APIs“ unter Anwendungsfall: Integration von Amazon Kinesis Data Streams mit der AWS Glue Schema Registry.

Entwickeln Sie Verbraucher mit AWS Lambda

Sie können eine AWS Lambda Funktion verwenden, um Datensätze in einem Datenstrom zu verarbeiten. AWS Lambda ist ein Rechendienst, mit dem Sie Code ausführen können, ohne Server bereitzustellen oder zu verwalten. Es führt Ihren Code nur bei Bedarf aus und skaliert automatisch – von einigen Anfragen pro Tag bis zu Tausenden pro Sekunde. Sie zahlen nur für die tatsächlich aufgewendete Zeit. Es werden keine Gebühren berechnet, solange Ihr Code nicht ausgeführt wird. Mit AWS Lambda können Sie Code für praktisch jede Art von Anwendung oder Back-End-Dienst ausführen, und das alles ohne Verwaltungsaufwand. Der Service führt Ihren Code auf einer hoch verfügbaren Datenverarbeitungsinfrastruktur aus and erledigt die gesamte Administration der Datenverarbeitungsressourcen, einschließlich der Server- und Betriebssystemwartung, Kapazitätsbereitstellung und automatischen Skalierung sowie Code-

Überwachung und -Protokollierung. Weitere Informationen finden Sie unter Verwenden AWS Lambda mit Amazon Kinesis.

Informationen zur Fehlerbehebung finden Sie unter Warum kann der Kinesis-Data-Streams-Trigger meine Lambda-Funktion nicht aufrufen?

Entwickeln Sie Kunden mithilfe von Amazon Managed Service für Apache Flink

Sie können eine Anwendung von Amazon Managed Service für Apache Flink verwenden, um Daten in einem Kinesis-Stream mit SQL, Java oder Scala zu verarbeiten und zu analysieren. Anwendungen des Managed Service für Apache Flink kann Daten mithilfe von Referenzquellen anreichern, Daten im Laufe der Zeit aggregieren oder Machine Learning verwenden, um Datenanomalien zu finden. Anschließend können Sie die Analyseergebnisse in einen anderen Kinesis-Stream, einen Firehose-Lieferstream oder eine Lambda-Funktion schreiben. Weitere Informationen finden Sie im Entwicklerhandbuch für Managed Service für Apache Flink für SQL-Anwendungen oder im Entwicklerhandbuch für Managed Service für Apache Flink für Flink-Anwendungen.

Entwickeln Sie Verbraucher mithilfe von Amazon Data Firehose

Sie können einen Firehose verwenden, um Datensätze aus einem Kinesis-Stream zu lesen und zu verarbeiten. Firehose ist ein vollständig verwalteter Service für die Bereitstellung von Echtzeit-Streaming-Daten an Ziele wie Amazon S3, Amazon Redshift, Amazon OpenSearch Service und Splunk. Firehose unterstützt auch alle benutzerdefinierten HTTP-Endpunkte oder HTTP-Endpunkte, die unterstützten Drittanbietern gehören, darunter Datadog, MongoDB und New Relic. Sie können Firehose auch so konfigurieren, dass Ihre Datensätze transformiert und das Datensatzformat konvertiert wird, bevor Ihre Daten an das Ziel gesendet werden. Weitere Informationen finden Sie unter Writing to Firehose Using Kinesis Data Streams.

Daten aus Kinesis Data Streams mithilfe anderer AWS Dienste lesen

Die folgenden AWS Services können direkt in Amazon Kinesis Data Streams integriert werden, um Daten aus Kinesis-Datenströmen zu lesen. Überprüfen Sie die Informationen für jeden Service, an dem Sie interessiert sind, und verweisen Sie auf die bereitgestellten Referenzen.

Themen

- Daten aus Kinesis Data Streams mit Amazon EMR lesen
- Daten aus Kinesis Data Streams mit Amazon EventBridge Pipes lesen
- Daten aus Kinesis Data Streams lesen mit AWS Glue
- Daten aus Kinesis Data Streams mit Amazon Redshift lesen

Daten aus Kinesis Data Streams mit Amazon EMR lesen

Amazon EMR-Cluster können Kinesis-Streams direkt lesen und verarbeiten, indem sie vertraute Tools aus dem Hadoop-Ökosystem wie Hive, Pig MapReduce, die Hadoop Streaming API und Cascading verwenden. Sie können auch Echtzeitdaten aus Kinesis Data Streams mit vorhandenen Daten auf Amazon S3, Amazon DynamoDB und HDFS in einem laufenden Cluster verbinden. Zur Nachbearbeitung können Sie die Daten aus Amazon EMR direkt in Amazon S3 oder DynamoDB laden.

Weitere Informationen finden Sie unter Amazon Kinesis im Handbuch zu Amazon-EMR-Releases.

Daten aus Kinesis Data Streams mit Amazon EventBridge Pipes lesen

Amazon EventBridge Pipes unterstützt Kinesis-Datenstreams als Quelle. Amazon EventBridge Pipes unterstützt Sie bei der Erstellung von point-to-point Integrationen zwischen Veranstaltern und Verbrauchern mit optionalen Transformations-, Filter- und Anreicherungsschritten. Sie können EventBridge Pipes verwenden, um Datensätze in einem Kinesis-Datenstream zu empfangen und diese Datensätze optional zu filtern oder zu verbessern, bevor Sie sie zur Verarbeitung an eines der verfügbaren Ziele senden, einschließlich Kinesis Data Streams.

Weitere Informationen finden Sie unter Amazon Kinesis Stream as a source im Amazon EventBridge Release Guide.

Daten aus Kinesis Data Streams lesen mit AWS Glue

Mit AWS Glue Streaming-ETL können Sie Streaming-Jobs zum Extrahieren, Transformieren und Laden (ETL) erstellen, die kontinuierlich ausgeführt werden und Daten aus Amazon Kinesis Data Streams verbrauchen. Die Aufträge bereinigen und transformieren die Daten und laden die Ergebnisse dann in Amazon-S3-Data-Lakes oder JDBC-Datenspeicher.

Weitere Informationen finden Sie unter Streaming-ETL-Aufträge in AWS Glue im AWS Glue - Release-Handbuch.

Daten aus Kinesis Data Streams mit Amazon Redshift lesen

Amazon Redshift unterstützt die Streaming-Erfassung von Amazon Kinesis Data Streams. Das Streaming-Erfassungs-Feature Amazon Redshift ermöglicht das Erfassen von Streaming-Daten mit geringer Latenz und hoher Geschwindigkeit aus Amazon Kinesis Data Streams in einer materialisierten Ansicht von Amazon Redshift. Durch die Amazon Redshift-Streaming-Aufnahme müssen Daten nicht mehr in Amazon S3 gespeichert werden, bevor sie in Amazon Redshift aufgenommen werden.

Weitere Informationen finden Sie unter Streaming-Erfassung im Handbuch zum Release von Amazon Redshift.

Lesen Sie mithilfe von Integrationen von Drittanbietern aus Kinesis Data Streams

Sie können Daten aus Amazon Kinesis Data Streams mit einer der folgenden Drittanbieteroptionen lesen, die in Kinesis Data Streams integriert sind. Wählen Sie die Option aus, über die Sie mehr erfahren möchten, und finden Sie Ressourcen und Links zu relevanter Dokumentation.

Themen

- Apache Flink
- Adobe Experience Platform
- Apache Druid
- Apache Spark
- Databricks
- Kafka Confluent Plattform
- Kinesumer
- Talend

Apache Flink

Apache Flink ist ein Framework und eine verteilte Verarbeitungs-Engine für statusbehaftete Berechnungen über unbegrenzte und begrenzte Datenströme. Weitere Informationen zur Nutzung

von Kinesis Data Streams mit Apache Flink finden Sie unter Amazon Kinesis Data Streams Connector.

Adobe Experience Platform

Die Adobe Experience Platform ermöglicht es Unternehmen, Kundendaten aus jedem System zu zentralisieren und zu standardisieren. Anschließend werden Datenwissenschaft und Machine Learning angewendet, um das Design und die Bereitstellung umfassender, personalisierter Erlebnisse erheblich zu verbessern. Weitere Informationen zur Nutzung von Kinesis-Datenströmen mit der Adobe Experience Platform finden Sie unter Amazon Kinesis Connector.

Apache Druid

Druid ist eine hochleistungsfähige Echtzeit-Analysedatenbank, die Abfragen auf Streaming- und Batch-Daten in Sekundenschnelle und bei hoher Auslastung ermöglicht. Weitere Informationen zur Aufnahme von Kinesis-Datenströmen mit Apache Druid finden Sie unter Amazon Kinesis-Aufnahme.

Apache Spark

Apache Spark ist eine einheitliche Analytics-Engine für die großflächige Datenverarbeitung. Es bietet hochwertige APIs Funktionen in Java, Scala, Python und R sowie eine optimierte Engine, die allgemeine Ausführungsdiagramme unterstützt. Sie können Apache Spark verwenden, um Anwendungen zur Stream-Verarbeitung zu erstellen, die die Daten in Ihren Kinesis-Datenströmen nutzen.

Verwenden Sie den Amazon Kinesis Data Streams-Connector, um Kinesis-Datenstreams mit Apache Spark Structured Streaming zu nutzen. Dieser Connector unterstützt die Nutzung mit Enhanced Fan-Out, wodurch Ihre Anwendung einen dedizierten Lesedurchsatz von bis zu 2 MB Daten pro Sekunde und Shard erhält. Weitere Informationen finden Sie unter Entwicklung benutzerdefinierter Verbraucher mit dediziertem Durchsatz (erweiterter Fan-Out).

Informationen zur Nutzung von Kinesis-Datenströmen mit Spark Streaming finden Sie unter Spark Streaming + Kinesis Integration.

Databricks

Databricks ist eine Cloud-basierte Plattform, die eine kollaborative Umgebung für Datentechnik, Datenwissenschaft und Machine Learning bietet. Weitere Informationen zur Nutzung von Kinesis-Datenströmen mithilfe von Databricks finden Sie unter Connect zu Amazon Kinesis herstellen.

Kafka Confluent Plattform

Confluent Platform basiert auf Kafka und bietet zusätzliche Features und Funktionen, mit denen Unternehmen Daten-Pipelines und Streaming-Anwendungen in Echtzeit aufbauen und verwalten können. Weitere Informationen zur Nutzung von Kinesis-Datenströmen mithilfe der Confluent Platform finden Sie unter Amazon Kinesis Source Connector for Confluent Platform.

Kinesumer

Kinesumer ist ein Go-Client, der einen clientseitigen Client für verteilte Verbrauchergruppen für Kinesis-Datenströme implementiert. Weitere Informationen finden Sie im Kinesumer-GitHub-Repository.

Talend

Talend ist eine Datenintegrations- und Verwaltungssoftware, die es Benutzern ermöglicht, Daten aus verschiedenen Quellen auf skalierbare und effiziente Weise zu sammeln, zu transformieren und miteinander zu verbinden. Weitere Informationen zur Nutzung von Kinesis-Datenströmen mit Talend finden Sie unter Talend mit einem Amazon Kinesis Kinesis-Stream Connect.

Problembehandlung bei Kinesis Data Streams Streams-Verbrauchern

Die folgenden Themen bieten Lösungen für häufig auftretende Probleme von Amazon Kinesis Data Streams Streams-Benutzern:

- Kompilierungsfehler mit dem Konstruktor LeaseManagementConfig
- Einige Kinesis Data Streams Streams-Datensätze werden bei der Verwendung der Kinesis Client Library übersprungen
- Datensätze, die zu demselben Shard gehören, werden gleichzeitig von verschiedenen Datensatzprozessoren verarbeitet
- Die Verbraucheranwendung liest langsamer als erwartet
- GetRecords gibt ein leeres Datensatz-Array zurück, auch wenn der Stream Daten enthält
- Der Shard-Iterator läuft unerwartet ab
- Die Verarbeitung von Verbraucherdaten hinkt hinterher

- Fehler beim Zugriff auf einen nicht autorisierten KMS-Schlüssel
- DynamoDbException: Der im Aktualisierungsausdruck angegebene Dokumentpfad ist für die Aktualisierung ungültig
- Beheben Sie andere häufig auftretende Probleme von Verbrauchern

Kompilierungsfehler mit dem Konstruktor LeaseManagementConfig

Beim Upgrade auf Kinesis Client Library (KCL) Version 3.x oder höher kann ein Kompilierungsfehler auftreten, der mit dem Konstruktor zusammenhängt. `LeaseManagementConfig` Wenn Sie direkt ein `LeaseManagementConfig` Objekt erstellen, um Konfigurationen festzulegen, anstatt es `ConfigsBuilder` in KCL-Versionen 3.x oder höher zu verwenden, wird Ihnen beim Kompilieren Ihres KCL-Anwendungscodes möglicherweise die folgende Fehlermeldung angezeigt.

```
Cannot resolve constructor 'LeaseManagementConfig(String, DynamoDbAsyncClient,
 KinesisAsyncClient, String)'
```

Bei KCL mit Versionen 3.x oder höher müssen Sie nach dem TableName-Parameter einen weiteren Parameter, applicationName (Typ: String), hinzufügen.

- Vorher: leaseManagementConfig = neu LeaseManagementConfig (TableName, Dynamo, KinesisClientDBClient, StreamName, WorkerIdentifier)
- Danach: leaseManagementConfig = neu LeaseManagementConfig (tableName, applicationName, Dynamo, KinesisClient, StreamNameDBClient, WorkerIdentifier)

Anstatt direkt ein LeaseManagementConfig Objekt zu erstellen, empfehlen `ConfigsBuilder` wir, Konfigurationen in KCL 3.x und späteren Versionen festzulegen. `ConfigsBuilder`bietet eine flexiblere und wartungsfreundlichere Methode zur Konfiguration Ihrer KCL-Anwendung.

Im Folgenden finden Sie ein Beispiel für die Verwendung `ConfigsBuilder` zum Festlegen von KCL-Konfigurationen.

```
ConfigsBuilder configsBuilder = new ConfigsBuilder(
    streamName,
    applicationName,
    kinesisClient,
    dynamoClient,
    cloudWatchClient,
    UUID.randomUUID().toString(),
```

```
    new SampleRecordProcessorFactory()
);

Scheduler scheduler = new Scheduler(
    configsBuilder.checkpointConfig(),
    configsBuilder.coordinatorConfig(),
    configsBuilder.leaseManagementConfig()
    .failoverTimeMillis(60000), // this is an example
    configsBuilder.lifecycleConfig(),
    configsBuilder.metricsConfig(),
    configsBuilder.processorConfig(),
    configsBuilder.retrievalConfig()
);
```

Einige Kinesis Data Streams Streams-Datensätze werden bei der Verwendung der Kinesis Client Library übersprungen

Der häufigste Grund für übersprungene Datensätzen ist ein Ausnahmefehler, der von `processRecords` ausgelöst wird. Die Kinesis Client Library (KCL) verlässt sich bei der Behandlung von Ausnahmen, die während der Verarbeitung von Datensätzen auftreten, auf Ihrem `processRecords`-Code. Alle von `processRecords` ausgelösten Ausnahmen werden von der KCL aufgenommen. Um unbegrenzte Wiederholungen eines Fehlers zu vermeiden, sendet die KCL den Datensatzstapel, der während der Ausnahme verarbeitet wurde, nicht erneut. Die KCL ruft stattdessen `processRecords` für den nächsten Datensatzstapel auf, ohne den Datensatzprozessor neu zu starten. Dies führt dazu, dass Konsumentenanwendungen übersprungene Datensätze erkennen. Zur Vermeidung übersprungener Datensätze müssen Sie alle Ausnahmen innerhalb von `processRecords` entsprechend behandeln.

Datensätze, die zu demselben Shard gehören, werden gleichzeitig von verschiedenen Datensatzprozessoren verarbeitet

Ein Shard hat für jede ausgeführte Anwendung der Kinesis Client Library (KCL) nur einen Besitzer. Allerdings kann es vorkommen, dass mehrere Datensatzprozessoren vorübergehend denselben Shard verarbeiten. Wenn eine Worker-Instance die Netzwerkkonnektivität verliert, geht die KCL davon aus, dass der nicht erreichbare Worker nach Ablauf der Failover-Zeit keine Datensätze mehr verarbeitet, und weist andere Worker-Instances an, die Kontrolle zu übernehmen. Für eine kurze Zeit werden dann möglicherweise Daten aus demselben Shard von neuen Datensatzprozessoren und Datensatzprozessoren des nicht erreichbaren Workers verarbeitet.

Stellen Sie eine Failover-Zeit ein, die für Ihre Anwendung geeignet ist. Für Anwendungen mit geringer Latenz entspricht der Standardwert von 10 Sekunden möglicherweise dem Zeitraum, den Sie maximal bereit sind zu warten. In den Fällen jedoch, in denen Sie Verbindungsprobleme erwarten, beispielsweise bei Aufrufen über verschiedene geografische Bereiche hinweg, in denen häufiger mit Verbindungsproblemen zu rechnen ist, ist die Zeitspanne möglicherweise zu gering.

Ihre Anwendung sollte diesem Szenario abhelfen können, insbesondere, weil die Netzwerkanbindung in der Regel für den zuvor nicht erreichbaren Worker wiederhergestellt wird. Wenn die Shards eines Datensatzprozessors von einem anderen Datensatzprozessor übernommen werden, müssen folgende Fälle für ein ordnungsgemäßes Herunterfahren korrekt behandelt werden:

1. Nach Abschluss des aktuellen Aufrufs von ruft die KCL die Shutdown-Methode auf dem Recordprozessor mit dem Shutdown-Grund 'ZOMBIE' auf. `processRecords` Von Ihren Datensatzprozessoren wird erwartet, dass alle Ressourcen entsprechend bereinigt werden und die Datensatzprozessoren anschließend beendet werden.
2. Wenn Sie versuchen, ein Checkpointing mit einem „Zombie“-Worker durchzuführen, löst die KCL `ShutdownException` aus. Nach Empfang der Ausnahme ist die aktuelle Methode durch Ihren Code sauber zu beenden.

Weitere Informationen finden Sie unter Behandeln Sie doppelte Datensätze.

Die Verbraucheranwendung liest langsamer als erwartet

Die häufigsten Gründe für einen langsameren Lesedurchsatz sind:

1. Bei mehreren Konsumentenanwendungen übersteigt die Anzahl der Lesevorgänge die pro Shard festgelegten Limits. Weitere Informationen finden Sie unter Kontingente und -Einschränkungen. Erhöhen Sie in diesem Fall die Anzahl der Shards im Kinesis-Datenstrom.
2. Das Limit für die maximale Anzahl von GetRecords pro Aufruf ist möglicherweise zu niedrig festgelegt. Wenn Sie die KCL verwenden, haben Sie möglicherweise einen niedrigen Wert bei der `maxRecords`-Eigenschaft des Workers angegeben. Grundsätzlich empfehlen wir die Verwendung der Systemvoreinstellungen für diese Eigenschaft.
3. Die Logik in Ihrem `processRecords` Aufruf kann aus verschiedenen Gründen länger als erwartet dauern. Die Logik ist möglicherweise CPU-intensiv, I/O blockierend oder führt zu Engpässen bei der Synchronisation. Führen Sie leere Datensatzprozessoren aus und vergleichen Sie den Lesedurchsatz, um festzustellen, ob dies der Fall ist. Weitere Informationen dazu, wie Sie

eingehende Daten zeitgerecht verarbeiten können, finden Sie unter Verwenden Sie Resharding, Skalierung und Parallelverarbeitung, um die Anzahl der Shards zu ändern.

Wenn Sie nur eine Konsumentenanwendung haben, ist es immer möglich, mindestens zwei Mal schneller als die PUT-Rate zu lesen. Das liegt daran, dass Sie bis zu 1 000 Datensätze pro Sekunde für Schreibvorgänge bis zu einem maximalen Datenschreibvolumen von 1 MB pro Sekunde (einschließlich Partitionsschlüsseln) schreiben können. Jeder offene Shard kann bis zu 5 Lesetransaktionen pro Sekunde unterstützen, bis zu einer maximalen Gesamtdatenleserate von 2 MB pro Sekunde. Beachten Sie, dass jeder Lesevorgang (GetRecords-Aufruf) einen Stapel an Datensätzen erhält. Die Menge der Daten, die von GetRecords zurückgegeben werden, hängt von der Shard-Auslastung ab. Die maximale Größe der Daten, die GetRecords zurückgeben kann, ist 10 MB. Wenn ein Anruf dieses Limit zurückgibt, lösen nachfolgende Aufrufe innerhalb der nächsten 5 Sekunden einen Fehler aus. `ProvisionedThroughputExceededException`

GetRecords gibt ein leeres Datensatz-Array zurück, auch wenn der Stream Daten enthält

Beim Verbrauchen oder Empfangen von Datensätzen kommt ein PULL-Modell zum Einsatz. Von Entwicklern wird erwartet, dass sie GetRecordsin einer Dauerschleife ohne Back-offs aufrufen. Jeder Aufruf von GetRecords gibt auch einen `ShardIterator`-Wert zurück, der in der nächsten Iteration der Schleife verwendet wird.

Die GetRecords-Operation blockiert nicht. Stattdessen erfolgt sofort eine Rückgabe, entweder mit relevanten Datensätzen oder mit einem leeren `Records`-Element. Ein leeres `Records`-Element wird unter zwei Bedingungen zurückgegeben:

1. Im Shard sind derzeit keine weiteren Daten vorhanden.
2. Es gibt keine Daten in der Nähe des Shard-Inhalts, auf den `ShardIterator` zeigt.

Die letzte Bedingung ist subtil, aber eine notwendige Design-Komponente, durch die unbegrenzte Suchzeiten (Latenzen) beim Abrufen von Datensätzen vermieden werden. Daher sollte die Anwendung, die Daten aus dem Stream verbraucht, eine Schleife durchlaufen, GetRecords aufrufen und leere Datensätze verarbeiten.

In einer Produktionsumgebung sollte die Endlosschleife nur verlassen werden, wenn der Wert von `NextShardIterator` `NULL` ist. Wenn `NextShardIterator` `NULL` ist, bedeutet dies, dass der aktuelle Shard geschlossen wurde und der Wert von `ShardIterator` ansonsten auf eine

Position hinter dem letzten Datensatz verweisen würde. Wenn die datenverbrauchende Anwendung SplitShard oder MergeShards nicht aufruft, bleibt der Shard offen und die Aufrufe von GetRecords geben niemals einen `NextShardIterator`-Wert von `NULL` zurück.

Wenn Sie die Kinesis Client Library (KCL) verwenden, wird das vorherige Verbrauchsmuster für Sie abstrahiert. Dies umfasst die automatische Handhabung einer Gruppe von Shards, die sich dynamisch ändert. Mit der KCL übermittelt der Entwickler nur die Logik für die Verarbeitung eingehender Datensätze. Dies ist möglich, da die Bibliothek GetRecords kontinuierlich für Sie aufruft.

Der Shard-Iterator läuft unerwartet ab

Bei jeder GetRecords-Anforderung (wie `NextShardIterator`) wird ein neuer Shard-Iterator zurückgegeben, den Sie bei der nächsten GetRecords-Anforderung (wie `ShardIterator`) verwenden. In der Regel läuft dieser Shard-Iterator nicht vor der Verwendung ab. Es kann jedoch sein, dass Shard-Iteratoren ihre Gültigkeit verlieren, weil Sie GetRecords länger als 5 Minuten nicht aufgerufen oder die Konsumentenanwendung neu gestartet haben.

Wenn der Shard-Iterator unmittelbar abläuft, bevor Sie ihn verwenden können, kann dies darauf hindeuten, dass die von Kinesis verwendete DynamoDB-Tabelle nicht über genügend Kapazität zum Speichern der Lease-Daten verfügt. Diese Situation tritt häufiger auf, wenn Sie viele Shards haben. Beheben Sie das Problem, indem Sie die Schreibkapazität der Shard-Tabelle erhöhen. Weitere Informationen finden Sie unter Verwenden Sie eine Leasetabelle, um nachzuverfolgen, welche Shards von der KCL-Consumer-Anwendung verarbeitet wurden.

Die Verarbeitung von Verbraucherdaten hinkt hinterher

Bei den meisten Anwendungsfällen lesen die Konsumentenanwendungen die neuesten Daten aus dem Stream. Unter bestimmten Umständen hängt das Lesen des Konsumenten hinterher. Dieses Verhalten möglicherweise nicht erwünscht. Sehen Sie sich die häufigsten Gründe für ein zu langsames Lesen des Datenkonsumenten an, wenn Sie ermittelt haben, wie weit die Lesevorgänge des Konsumenten hinterherhängen.

Beginnen Sie mit der `GetRecords.IteratorAgeMilliseconds`-Metrik. Diese verfolgt die Leseposition aller Shards und Konsumenten im Stream. Beachten Sie: Wenn das Alter eines Iterators 50 % des Aufbewahrungszeitraums (standardmäßig 24 Stunden, anpassbar auf bis zu 365 Tage) überschreitet, besteht die Gefahr des Datenverlusts durch Datensatzablauf. Als schnelle Übergangslösung eignet sich eine Verlängerung des Aufbewahrungszeitraums. Auf diese Weise wird der Verlust von wichtigen Daten gestoppt, während Sie sich der weiteren Fehlerbehebung widmen.

Weitere Informationen finden Sie unter Überwachen Sie den Amazon Kinesis Data Streams Streams-Service mit Amazon CloudWatch. Identifizieren Sie als Nächstes anhand einer benutzerdefinierten CloudWatch Metrik, die von der Kinesis Client Library (KCL) ausgegeben wird, wie weit Ihre Verbraucheranwendung von jedem Shard liest. `MillisBehindLatest` Weitere Informationen finden Sie unter Überwachen Sie die Kinesis-Clientbibliothek mit Amazon CloudWatch.

Im Folgenden sind die häufigsten Gründe für ein Hinterherhängen von Konsumenten aufgeführt:

- Plötzliche Anstiege auf `GetRecords.IteratorAgeMilliseconds` oder `MillisBehindLatest` weisen in der Regel auf ein vorübergehendes Problem hin, beispielsweise Fehler bei einer API-Operation in einer nachgelagerten Anwendung. Untersuchen Sie diese plötzlichen Erhöhungen, wenn eine der Metriken dieses Verhalten konsistent zeigt.
- Eine allmählicher Anstieg dieser Metriken deutet darauf hin, dass ein Konsument Datensätze nicht schnell genug verarbeitet. Die häufigsten Ursachen für ein solches Verhalten sind unzureichende physische Ressourcen oder eine Datensatzverarbeitungslogik, die nicht für einen Anstieg des Stream-Durchsatzes skaliert wurde. Sie können dieses Verhalten überprüfen, indem Sie sich die anderen benutzerdefinierten CloudWatch Messwerte ansehen, die die KCL im Zusammenhang mit dem `processTask` Vorgang ausgibt, einschließlich `RecordProcessor.processRecords.TimeSuccess`, und. `RecordsProcessed`
  - Wenn Sie eine Erhöhung in der `processRecords.Time`-Metrik feststellen, die mit einem Anstieg des Durchsatzes korreliert, sollten Sie Ihre Datensatzverarbeitungslogik analysieren, um herauszufinden, warum sich diese nicht an den erhöhten Durchsatz anpasst.
  - Wenn Sie eine Erhöhung der `processRecords.Time`-Werte feststellen, die nicht in Zusammenhang mit einem erhöhten Durchsatz steht, prüfen Sie, ob im kritischen Pfad blockierende Aufrufe durchgeführt werden. Diese sind häufig der Grund für eine verlangsamte Datensatzverarbeitung. Alternativ können Sie auch die Parallelität durch eine Erhöhung der Anzahl der Shards erhöhen. Stellen Sie abschließend sicher, dass Sie bei Spitzenauslastung über eine ausreichende Menge an physischen Ressourcen (Arbeitsspeicher, CPU-Auslastung usw.) auf den zugrunde liegenden Verarbeitungsknoten verfügen.

Fehler beim Zugriff auf einen nicht autorisierten KMS-Schlüssel

Dieser Fehler tritt auf, wenn eine Verbraucheranwendung ohne Berechtigungen für den AWS KMS Schlüssel aus einem verschlüsselten Stream liest. Informationen zum Zuweisen von Berechtigungen zu einer Anwendung für den Zugriff auf einen KMS-Schlüssel finden Sie unter Verwenden von Schlüsselrichtlinien in AWS KMS und Verwenden von IAM-Richtlinien mit AWS KMS.

DynamoDbException: Der im Aktualisierungsausdruck angegebene Dokumentpfad ist für die Aktualisierung ungültig

Wenn Sie KCL 3.x mit den AWS SDK für Java Versionen 2.27.19 bis 2.27.23 verwenden, kann die folgende DynamoDB-Ausnahme auftreten:

„software.amazon.awssdk.services.dynamodb.model. DynamoDbException: Der im Aktualisierungsausdruck angegebene Dokumentpfad ist für die Aktualisierung ungültig (Service:, Statuscode: 400, Anforderungs-ID: xxx)“ DynamoDb

Dieser Fehler tritt aufgrund eines bekannten Problems in der auf AWS SDK für Java , das die von KCL 3.x verwaltete DynamoDB-Metadatentabelle betrifft. Das Problem wurde in Version 2.27.19 eingeführt und betrifft alle Versionen bis 2.27.23. Das Problem wurde in Version 2.27.24 behoben. AWS SDK für Java Für optimale Leistung und Stabilität empfehlen wir ein Upgrade auf Version 2.28.0 oder höher.

Beheben Sie andere häufig auftretende Probleme von Verbrauchern

- Warum kann der Kinesis-Data-Streams-Trigger meine Lambda-Funktion nicht aufrufen?
- Wie erkenne und behebe ich ReadProvisionedThroughputExceeded Ausnahmen in Kinesis Data Streams?
- Warum habe ich Probleme mit hoher Latenz bei Kinesis Data Streams?
- Warum gibt mein Kinesis-Datenstrom einen 500 Internal Server Error zurück?
- Wie behebe ich Fehler bei einer blockierten oder hängengebliebenen KCL-Anwendung für Kinesis Data Streams?
- Kann ich verschiedene Anwendungen der Amazon Kinesis Client Library mit der gleichen Amazon-DynamoDB-Tabelle verwenden?

Optimieren Sie die Verbraucher von Amazon Kinesis Data Streams

Sie können Ihren Amazon Kinesis Data Streams Streams-Nutzer auf der Grundlage des spezifischen Verhaltens, das Sie beobachten, weiter optimieren.

Sehen Sie sich die folgenden Themen an, um Lösungen zu finden.

Topics

- Verbessern Sie die Verarbeitung mit niedriger Latenz

- Verarbeiten Sie serialisierte Daten AWS Lambda mithilfe der Amazon Kinesis Producer Library
- Verwenden Sie Resharding, Skalierung und Parallelverarbeitung, um die Anzahl der Shards zu ändern
- Behandeln Sie doppelte Datensätze
- Kümmere dich um das Starten, Herunterfahren und Drosseln

Verbessern Sie die Verarbeitung mit niedriger Latenz

Die Übertragungsverzögerung ist definiert als die end-to-end Latenz von dem Moment, in dem ein Datensatz in den Stream geschrieben wird, bis zu seinem Lesen durch eine Verbraucheranwendung. Diese Verzögerung variiert und hängt von vielen Faktoren ab, im Wesentlichen aber vom Abrufintervall der Konsumentenanwendungen.

Für die meisten Anwendungen empfehlen wir, jeden Shard pro Anwendung einmal pro Sekunde abzurufen. So können mehrere Konsumentenanwendungen einen Stream gleichzeitig unabhängig voneinander verarbeiten, ohne an die Limits von 5 `GetRecords`-Aufrufen pro Sekunde von Amazon Kinesis Data Streams zu stoßen. Darüber hinaus ist die Verarbeitung größerer Datenstapel in Bezug auf Netzwerk- und andere nachgelagerte Latenzzeiten effizienter.

Die Standardeinstellungen der KCL entsprechend der bewährten Einstellung von einem Abruf pro Sekunde. Dieser Standard führt zu einer durchschnittlichen Verbreitungsverzögerung von weniger als 1 Sekunde.

Die Datensätze von Kinesis Data Streams können sofort nach dem Schreiben gelesen werden. In Fällen, in denen Daten sofort nach Bereitstellung verarbeitet werden müssen, ist dies eine wichtiger Aspekt. Sie können die Verbreitungsverzögerung erheblich verringern, indem Sie die Standardeinstellungen der KCL überschreiben, um häufiger Datensätze abzurufen. Sehen Sie sich dazu die folgenden Beispiele an.

Java KCL-Konfigurationscode:

```
kinesisClientLibConfiguration = new
        KinesisClientLibConfiguration(applicationName,
        streamName,
        credentialsProvider,

 workerId).withInitialPositionInStream(initialPositionInStream).withIdleTimeBetweenReadsInMilli
```

Einstellung der Eigenschaftendatei für Python- und Ruby-KCL:

```
idleTimeBetweenReadsInMillis = 250
```

> **Note**
>
> Da für Kinesis Data Streams ein Limit von 5 `GetRecords`-Aufrufen pro Sekunde pro Shard gilt, kann das Festlegen der `idleTimeBetweenReadsInMillis`-Eigenschaft auf einen Wert von unter 200 ms dazu führen, dass Ihre Anwendung die `ProvisionedThroughputExceededException`-Ausnahme erkennt. Zu viele dieser Ausnahmen können zu exponentiellen Backoffs führen und bedeutsame unerwartete Latenzzeiten bei der Verarbeitung nach sich ziehen. Wenn Sie diese Eigenschaft auf 200 ms oder etwas mehr setzen und mehr als eine Verarbeitungsanwendung haben, kommt es zu einer ähnlichen Drosselung.

Verarbeiten Sie serialisierte Daten AWS Lambda mithilfe der Amazon Kinesis Producer Library

Die Amazon Kinesis Producer Library (KPL) fasst kleine benutzerformatierte Datensätze zu größeren Datensätzen von bis zu 1 MB zusammen, um den Durchsatz von Amazon Kinesis Data Streams besser zu nutzen. Die KCL für Java unterstützt zwar die Deaggregation dieser Datensätze, Sie müssen jedoch ein spezielles Modul verwenden, um Datensätze zu deaggregieren, wenn Sie sie als Verbraucher Ihrer Streams verwenden. AWS Lambda Den erforderlichen Projektcode und Anweisungen finden Sie GitHub unter Amazon Kinesis Producer Library Deaggregation Modules for Lambda. AWS Die Komponenten in diesem Projekt bieten Ihnen die Möglichkeit, KPL-serialisierte Daten in Java AWS Lambda, Node.js und Python zu verarbeiten. Diese Komponenten können auch als Teil einer mehrsprachigen KCL-Anwendung verwendet werden.

Verwenden Sie Resharding, Skalierung und Parallelverarbeitung, um die Anzahl der Shards zu ändern

Das Resharding ermöglicht Ihnen das Erhöhen oder Verringern der Anzahl der Shards in einem Stream zur Anpassung an die Datenrate. Das Resharding wird üblicherweise von einer administrativen Anwendung durchgeführt, die Datenverwaltungsmetriken zu Shards überwacht. Obwohl die KCL selbst keine Resharding-Vorgänge auslöst, kann sie sich an die geänderte Anzahl von Shards anpassen, die sich aus dem Resharding ergibt.

Wie unter Verwenden Sie eine Leasetabelle, um nachzuverfolgen, welche Shards von der KCL-Consumer-Anwendung verarbeitet wurden erwähnt, verfolgt die KCL die Shards im Stream mit einer Amazon-DynamoDB-Tabelle. Wenn aufgrund einer Resharding-Operation neue Shards erstellt werden, werden diese von der KCL erkannt, die dann neue Zeilen in die Tabelle einfügt. Die Auftragnehmer entdecken die neuen Shards automatisch und erstellen Prozessoren, um die Daten dieser neuen Shards zu verwalten. Die KCL verteilt die Shards zudem auf alle verfügbaren Auftragnehmer und Datensatzprozessoren im Stream.

Die KCL stellt sicher, dass alle Daten, die vor dem Resharding bereits in einem Shard vorhanden waren, als erstes verarbeitet werden. Nachdem diese Daten verarbeitet wurden, werden die Daten der neuen Shards an die Datensatzprozessoren gesendet. So behält die KCL die Reihenfolge bei, in der die Datensätze für einen bestimmten Partitionsschlüssel zum Stream hinzugefügt wurden.

Beispiel: Resharding, Skalierung und Parallelverarbeitung

Das folgende Beispiel veranschaulicht, wie Sie die KCL bei der Skalierung und beim Resharding unterstützt:

- Angenommen, Ihre Anwendung wird auf einer EC2 Instance ausgeführt und verarbeitet einen Kinesis-Datenstrom mit vier Shards. Diese eine Instance verfügt über einen KCL-Auftragnehmer und vier Datensatzprozessoren (ein Datensatzprozessor für jeden Shard). Diese vier Datensatzprozessoren werden parallel innerhalb desselben Prozesses ausgeführt.
- Wenn Sie dann die Anwendung für die Verwendung einer weiteren Instance skalieren, verarbeiten zwei Instances einen Stream mit vier Shards. Wenn der KCL-Auftragnehmer auf der zweiten Instance gestartet wird, führt er einen Lastausgleich mit der ersten Instance durch, sodass nun jede Instance zwei Shards verarbeitet.
- Dann beschließen Sie, Ihre vier Shards auf fünf Shards aufzuteilen. Die KCL koordiniert erneut die Verarbeitung auf den Instances: Eine Instance verarbeitet drei Shards und die andere zwei. Eine ähnliche Koordinierung erfolgt, wenn Sie Shards zusammenführen.

In der Regel sollten Sie bei Verwendung der KCL sicherstellen, dass die Anzahl der Instances nicht die Anzahl der Shards übersteigt (außer für Standby-Zwecke im Falle eines Ausfalls). Jeder Shard wird exakt von einem KCL-Auftragnehmer verarbeitet und verfügt über genau einen entsprechenden Datensatzprozessor, deshalb benötigen Sie für die Verarbeitung eines Shards niemals mehrere Instances. Ein Auftragnehmer kann allerdings eine beliebige Anzahl von Shards verarbeiten, deshalb ist es kein Problem, wenn Sie mehr Shards als Instances haben.

Testen Sie eine Kombination der folgenden Ansätze, wenn Sie die Verarbeitung in Ihrer Anwendung optimieren möchten:

- Erhöhung der Instance-Größe (da alle Datensatzprozessoren innerhalb eines Prozesses parallel ausgeführt werden)
- Erhöhung der Anzahl der Instances bis zur maximalen Anzahl offener Shards (da Shards unabhängig verarbeitet werden können)
- Erhöhung der Anzahl der Shards (fördert die Parallelität)

Beachten Sie, dass Sie Auto Scaling für die automatische Skalierung Ihrer Instances basierend auf entsprechenden Metriken nutzen können. Weitere Informationen hierzu finden Sie im Amazon EC2 Auto Scaling-Benutzerhandbuch.

Wenn durch das Resharding die Anzahl der Shards im Stream erhöht wird, führt eine entsprechende Erhöhung der Datensatzprozessoren zu einer höheren Last der EC2 Instances, auf denen sie gehostet werden. Wenn die Instances Teil einer Auto Scaling-Gruppe sind und die Last entsprechend steigt, fügt die Auto Scaling-Gruppe weitere Instances hinzu, damit die höhere Last ausgeglichen wird. Konfigurieren Sie Ihre Instances so, dass Ihre Anwendung Amazon Kinesis Data Streams beim Startup gestartet wird. So werden zusätzliche Auftragnehmer und Datensatzprozessoren in der neuen Instance sofort aktiviert.

Weitere Informationen zum Resharding finden Sie unter Einen Stream erneut teilen.

Behandeln Sie doppelte Datensätze

Es gibt zwei primäre Gründe, warum Datensätze mehr als einmal an Ihre Anwendung von Amazon Kinesis Data Streams übermittelt werden: Wiederholungsversuche des Produzenten und Wiederholungsversuche des Konsumenten. Ihre Anwendung muss in der Lage sein, einzelne Datensätze mehrere Male angemessen zu verarbeiten.

Der Produzent versucht es erneut

Stellen Sie sich einen Produzenten vor, der nach dem Aufruf von `PutRecord` aber noch vor dem Erhalt einer Bestätigung von Amazon Kinesis Data Streams einen netzwerkbedingten Timeout erfährt. In diesem Fall ist es fraglich, ob der Datensatz auch an Kinesis Data Streams übermittelt wurde. Davon ausgehend, dass jeder Datensatz für die Anwendung wichtig ist, ist der Produzent so aufgebaut, dass er den Aufruf mit denselben Daten wiederholt. Wenn beide `PutRecord`-Aufrufe

mit denselben Daten erfolgreich an Kinesis Data Streams übergeben werden, sind anschließend zwei Datensätze von Kinesis Data Streams vorhanden. Obwohl die beiden Datensätze identische Daten enthalten, haben Sie trotzdem eindeutige Sequenznummern. Anwendungen, die Garantien benötigen, sollten einen Primärschlüssel in den Datensatz einbetten, sodass doppelte Datensätze später bei der Verarbeitung entfernt werden. Beachten Sie, dass die Anzahl der Duplikate aufgrund von Wiederholungen durch den Produzenten verglichen mit der Anzahl der Duplikate aufgrund von Wiederholungen durch den Konsumenten in der Regel nicht sehr hoch ist.

> **Note**
>
> Wenn Sie das AWS SDK verwenden`PutRecord`, erfahren Sie mehr über das SDK-Wiederholungsverhalten im AWS SDKs und Tools-Benutzerhandbuch.

Der Verbraucher versucht es erneut

Wiederholungen durch den Konsumenten (Datenverarbeitungsanwendungen) kommen vor, wenn Datenprozessoren neu gestartet werden. Datensatzprozessoren für denselben Shard werden in den folgenden Fällen neu gestartet:

1. Ein Auftragnehmer wird unerwartet beendet
2. Es werden Auftragnehmer-Instances hinzugefügt oder entfernt
3. Es werden Shards zusammengeführt oder getrennt
4. Die Anwendung wird bereitgestellt

In all diesen Fällen wird die shards-to-worker-to -record-processor-Zuordnung kontinuierlich aktualisiert, um die Lastenausgleichsverarbeitung zu berücksichtigen. Shard-Prozessoren, die in andere Instances migriert wurden, starten das Verarbeiten von Datensätzen am letzten Prüfpunkt neu. Dies führt zu einer doppelten Datensatzverarbeitung, wie in dem folgenden Beispiel dargestellt. Weitere Informationen zur Lastverteilung finden Sie unter Verwenden Sie Resharding, Skalierung und Parallelverarbeitung, um die Anzahl der Shards zu ändern.

Beispiel: Wiederholte Versuche eines Verbrauchers führen zu erneut zugestellten Datensätzen

In diesem Beispiel haben Sie eine Anwendung, die kontinuierlich Datensätze aus einem Stream liest, Datensätze in eine lokale Datei aggregiert und die Datei in Amazon S3 hochlädt. Der Einfachheit halber gehen wir davon aus, dass nur 1 Shard und 1 Auftragnehmer den Shard verarbeiten.

Beachten Sie die folgende Beispielabfolge von Ereignissen und gehen Sie davon aus, dass der letzte Prüfpunkt bei Datensatznummer 10.000 gesetzt wurde.

1. Ein Auftragnehmer liest den nächsten Datensatzstapel aus dem Shard (Datensätze 10.001 bis 20.000).
2. Anschließend übermittelt der Auftragnehmer diesen an den zugehörigen Datensatzprozessor.
3. Der Datensatzprozessor aggregiert die Daten, erstellt eine Amazon-S3-Datei und lädt die Datei erfolgreich in Amazon S3 hoch.
4. Der Auftragnehmer wird unerwartet beendet, noch ehe er einen neuen Prüfpunkt setzen kann.
5. Anwendung, Auftragnehmer und Datensatzprozessor starten neu.
6. Der Auftragnehmer beginnt beim letzten erfolgreich gesetzten Prüfpunkt mit dem Lesen, hier Datensatznummer 10.001.

Somit werden die Datensätze 10.001 bis 20.000 mehr als einmal verwendet.

Widerstandsfähigkeit gegenüber Wiederholungsversuchen durch Verbraucher

Auch wenn Datensätze mehrmals verarbeitet werden, kann die Anwendung vorgeben, dass Datensätze nur einmal verarbeitet wurden (idempotente Verarbeitung). Die Lösungen für dieses Problem variieren in Komplexität und Genauigkeit. Wenn das Ziel der resultierenden Daten doppelte Datensätze ordnungsgemäß handhaben kann, sollten Sie sich auf diese Anwendung verlassen, um eine idempotente Verarbeitung zu erreichen. Mit Opensearch können Sie beispielsweise eine Kombination aus Versionierung und Unique verwenden, um doppelte Verarbeitung IDs zu verhindern.

Die Beispielanwendung aus dem vorherigen Abschnitt liest kontinuierlich Datensätze aus dem Stream, aggregiert diese in eine lokale Datei und lädt diese Datei in Amazon S3 hoch. Wie gezeigt, werden die Datensätze 10 001 bis 20 000 mehr als einmal verarbeitet, sodass es mehrere Amazon-S3-Dateien mit identischen Daten gibt. Eine Möglichkeit, Duplikate in diesem Beispiel zu vermeiden, besteht darin, sicherzustellen, dass bei Schritt 3 das folgende Schema verwendet wird:

1. Der Datensatzprozessor verwendet eine feste Anzahl von Datensätzen pro Amazon-S3-Datei, beispielsweise 5 000.
2. Der Dateiname nutzt folgendes Schema: Amazon-S3-Präfix, Shard-ID und `First-Sequence-Num`. In diesem Fall könnte dies `sample-shard000001-10001` sein.
3. Nach dem Hochladen der Amazon-S3-Datei setzen Sie durch Angabe von `Last-Sequence-Num` einen Prüfpunkt. In diesem Fall würden Sie bei Datensatznummer 15.000 einen Prüfpunkt setzen.

Bei diesem Schema hat die resultierende Amazon-S3-Datei auch bei einer mehrfachen Verarbeitung der Datensätze denselben Namen und Dateninhalt. Die Wiederholungen führen nur dazu, dass dieselben Daten mehrmals in dieselbe Datei geschrieben werden.

Bei einer Reshard-Operation ist die Anzahl der im Shard verbliebenen Datensätze möglicherweise kleiner als benötigte Anzahl Ihrer gewünschten, festen Anzahl. In diesem Fall muss die `shutdown()`-Methode die Datei an Amazon S3 übertragen und bei der letzten Sequenznummer einen Prüfpunkt setzen. Die obige Schema ist auch mit Reshard-Operationen kompatibel.

Kümmere dich um das Starten, Herunterfahren und Drosseln

Nachfolgend einige Überlegungen, die Sie in das Design Ihrer Amazon Kinesis Data Streams einfließen lassen sollten.

Topics

- Starten Sie Datenproduzenten und Datenkonsumenten
- Fahren Sie eine Amazon Kinesis Data Streams Streams-Anwendung herunter
- Drosselung von Lesevorgängen

Starten Sie Datenproduzenten und Datenkonsumenten

Standardmäßig beginnt die KCL am vorderen Ende des Streams mit dem Lesen von Datensätzen, also mit dem zuletzt hinzugefügten Datensatz. Wenn bei dieser Konfiguration eine datenproduzierende Anwendung Datensätze zum Stream hinzufügt, ehe empfangsbereite Datensatzprozessoren ausgeführt werden, werden diese Datensätze nach dem Start der Datensatzprozessoren nicht gelesen.

Damit Datenprozessoren immer am Anfang des Streams mit dem Lesen beginnen, legen Sie den folgenden Wert in der Eigenschaftendatei Ihrer Amazon Kinesis Data Streams fest:

```
initialPositionInStream = TRIM_HORIZON
```

Standardmäßig speichert Amazon Kinesis Data Streams alle Daten für 24 Stunden. Er unterstützt auch eine erweiterte Aufbewahrung von bis zu 7 Tagen und die langfristige Aufbewahrung von bis zu 365 Tagen. Dieser Zeitrahmen wird als Aufbewahrungszeitraum bezeichnet. Wenn Sie die Startposition auf `TRIM_HORIZON` festsetzen, beginnt der Datensatzprozessor entsprechend des definierten Aufbewahrungszeitraums mit den ältesten Daten. Wenn ein Datensatzprozessor nach Ablauf des Aufbewahrungszeitraums gestartet wird, sind trotz `TRIM_HORIZON`-Einstellung

einige der Datensätze aus dem Stream nicht mehr verfügbar. Aus diesem Grund sollten Verbraucheranwendungen immer aus dem Stream lesen und anhand der CloudWatch Metrik überwachen, `GetRecords.IteratorAgeMilliseconds` ob die Anwendungen mit den eingehenden Daten Schritt halten.

In einigen Fällen mag es unerheblich sein, wenn Datensatzprozessoren die ersten Datensätze im Stream auslassen. Sie könnten beispielsweise einige erste Datensätze durch den Stream laufen lassen, um zu testen, ob der Stream end-to-end erwartungsgemäß funktioniert. Nach dieser anfänglichen Verifizierung starten Sie dann die Auftragnehmer und beginnen mit der Einleitung von Produktionsdaten in den Stream.

Weitere Informationen zur `TRIM_HORIZON`-Einstellung finden Sie unter Verwenden Sie Shard-Iteratoren.

Fahren Sie eine Amazon Kinesis Data Streams Streams-Anwendung herunter

Wenn Ihre Anwendung für Amazon Kinesis Data Streams die beabsichtigten Aufgaben abgeschlossen hat, sollten Sie sie herunterfahren, indem Sie die EC2-Instances beenden, auf denen sie ausgeführt wird. Sie können die Instances über die AWS-Managementkonsole oder die AWS CLI beenden.

Nach dem Herunterfahren der Anwendung für Amazon Kinesis Data Streams sollten Sie die Amazon-DynamoDB-Tabelle löschen, mit der die KCL den Anwendungsstatus verfolgt hat.

Drosselung von Lesevorgängen

Der Durchsatz eines Streams wird auf Shard-Ebene bereitgestellt. Jeder Shard hat einen Lesedurchsatz von bis zu 5 Transaktionen pro Sekunde für Lesevorgänge, bis zu einer maximalen Gesamtdatenleserate von 2 MB pro Sekunde. Wenn eine Anwendung (oder eine Gruppe von Anwendungen im selben Stream) versucht, Daten schneller aus einem Shard abzurufen, drosselt Kinesis Data Streams die entsprechenden GET-Operationen.

Wenn in einer Amazon Kinesis Data Streams ein Datensatzprozessor Daten schneller als das Limit verarbeitet, beispielsweise bei einem Ausfall, kommt es zu einem Failover. Da die KCL die Interaktionen zwischen der Anwendung und Kinesis Data Streams verwaltet, treten Ablehnungsausnahmen im Code der KCL und nicht im Anwendungscode auf. Da die KCL diese Ausnahmen jedoch protokolliert, sind sie in den Protokollen zu sehen.

Wenn Sie feststellen, dass Ihre Anwendung permanent gedrosselt wird, sollten Sie eine Erhöhung der Shards für diesen Stream in Betracht ziehen.

Kinesis Data Streams überwachen

Sie können Datenströme in Amazon Kinesis Data Streams mit den folgenden Features überwachen:

- CloudWatch Metriken — Kinesis Data Streams sendet CloudWatch benutzerdefinierte Amazon-Metriken mit detaillierter Überwachung für jeden Stream.
- Kinesis Agent — Der Kinesis Agent veröffentlicht benutzerdefinierte CloudWatch Metriken, anhand derer beurteilt werden kann, ob der Agent erwartungsgemäß funktioniert.
- API-Protokollierung – Kinesis Data Streams verwendet AWS CloudTrail zur Protokollierung von API-Aufrufen und speichert die Daten in einem Amazon-S3-Bucket.
- Kinesis Client Library – Die Kinesis Client Library (KCL) bietet Metriken pro Shard, Arbeiter und KCL-Anwendung.
- Kinesis Producer Library — Die Amazon Kinesis Producer Library (KPL) bietet Metriken pro Shard, Worker und KPL-Anwendung.

Weitere Informationen zu allgemeinen Überwachungsproblemen, Fragen und zur Fehlerbehebung finden Sie im Folgenden:

- Welche Metriken sollte ich verwenden, um Probleme mit Kinesis Data Streams zu überwachen und zu beheben?
- Warum steigt der IteratorAgeMilliseconds Wert von Kinesis Data Streams ständig?

Überwachen Sie den Amazon Kinesis Data Streams Streams-Service mit Amazon CloudWatch

Amazon Kinesis Data Streams und Amazon CloudWatch sind integriert, sodass Sie CloudWatch Metriken für Ihre Kinesis-Datenstreams sammeln, anzeigen und analysieren können. Um beispielsweise die Shard-Nutzung zu verfolgen, können Sie die Kennzahlen `IncomingBytes` und `OutgoingBytes` überwachen und mit der Anzahl der Shards im Stream vergleichen.

Stream-Metriken und Metriken auf Shard-Ebene, die Sie konfigurieren, werden automatisch erfasst und in jede Minute übertragen. CloudWatch Die Metriken werden für zwei Wochen archiviert. Nach Ablauf dieses Zeitraums werden die Daten verworfen.

Die folgende Tabelle beschreibt die Überwachung von Kinesis-Datenströmen auf grundlegender Stream- und erweiterter Shard-Ebene.

Typ	Description
Grundlegend (Stream-Ebene)	Daten auf Stream-Ebene werden automatisch und kostenlos jede Minute gesendet.
Erweitert (Shard-Ebene)	Daten auf Shard-Ebene werden für zusätzlic he Kosten jede Minute gesendet. Für diese Datenebene müssen Sie eine spezielle Aktivierung für den Stream mit der Operation EnableEnhancedMonitoring vornehmen. Informationen zu den Preisen finden Sie auf der CloudWatch Amazon-Produktseite.

Dimensionen und Metriken von Amazon Kinesis Data Streams

Kinesis Data Streams sendet Metriken CloudWatch auf zwei Ebenen: auf Stream-Ebene und optional auf Shard-Ebene. Metriken auf Stream-Ebene eignen sich für die häufigsten Anwendungsfälle zur Überwachung unter normalen Bedingungen. Metriken auf Shard-Ebene sind für bestimmte Überwachungsaufgaben gedacht, gewöhnlich im Zusammenhang mit der Fehlerbehebung; sie werden normalerweise mit der Operation EnableEnhancedMonitoring aktiviert.

Eine Erläuterung der CloudWatch anhand von Metriken gesammelten Statistiken finden Sie unter CloudWatch Statistiken im CloudWatch Amazon-Benutzerhandbuch.

Themen

- Grundlegende Metriken auf Stream-Ebene
- Verbesserte Metriken auf Shard-Ebene
- Dimensionen für Amazon Kinesis Data Streams Streams-Metriken
- Empfohlene Amazon Kinesis Data Streams Streams-Metriken

Grundlegende Metriken auf Stream-Ebene

Der `AWS/Kinesis`-Namespace enthält die folgenden Metriken auf Stream-Ebene.

Kinesis Data Streams sendet diese Metriken auf Stream-Ebene an CloudWatch jede Minute. Diese Metriken sind jederzeit verfügbar:

Metrik	Description
`GetRecords.Bytes`	Anzahl der Byte, die im angegebenen Zeitraum vom Kinesis-Stream abgerufen wurden. Die Statistiken Minimum, Maximum und Durchschnitt geben die Byte in einem einzelnen `GetRecords` -Vorgang für den Stream im angegebenen Zeitraum an. Metrikname auf Shard-Ebene: `OutgoingBytes` Abmessungen: StreamName Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben Einheiten: Byte
`GetRecords.IteratorAge`	Diese Metrik wird nicht mehr verwendet. Verwenden Sie `GetRecords.IteratorAgeMilliseconds` .
`GetRecords.IteratorAgeMilliseconds`	Alter des letzten Datensatzes in allen `GetRecords` -Aufrufen an einen Kinesis-Stream, gemessen im angegebenen Zeitraum. Das Alter ist die Differenz zwischen der aktuellen Zeit und der Zeit, zu der der letzte Datensatz des `GetRecords` -Aufrufs in den Stream geschrieben wurde. Die Statistiken Minimum und Maximum können zum Nachverfolgen des Fortschritts von Kinesis-Verbraucheranwendungen verwendet werden. Wenn der Wert null beträgt, sind die gelesenen Datensätze vollständig mit dem Stream aktualisiert. Metrikname auf Shard-Ebene: `IteratorAgeMilliseconds` Abmessungen: StreamName

Metrik	Description
	Gültige Statistiken: Minimum, Maximum, Durchschnitt, Stichproben Einheiten: Millisekunden
`GetRecords.Latency`	Die Dauer pro `GetRecords` -Vorgang, gemessen im angegebenen Zeitraum. Abmessungen: StreamName Statistiken: Minimum, Maximum, Durchschnitt Einheiten: Millisekunden
`GetRecords.Records`	Anzahl der Datensätze, die im angegebenen Zeitraum vom Shard abgerufen wurden. Die Statistiken Minimum, Maximum und Durchschnitt geben die Datensätze in einem einzelnen `GetRecords` -Vorgang für den Stream im angegebenen Zeitraum an. Metrikname auf Shard-Ebene: `OutgoingRecords` Abmessungen: StreamName Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben Einheiten: Anzahl
`GetRecords.Success`	Anzahl der erfolgreichen `GetRecords` -Vorgänge pro Stream im angegebenen Zeitraum. Abmessungen: StreamName Gültige Statistiken: Durchschnitt, Summe, Stichproben Einheiten: Anzahl

Metrik	Description
`IncomingBytes`	Anzahl der Byte, die im angegebenen Zeitraum erfolgreich an den Kinesis-Stream übertragen wurden. Diese Metrik enthält die Byte von `PutRecord` - und `PutRecords` -Vorgängen. Die Statistiken Minimum, Maximum und Durchschnitt geben die Byte in einem einzelnen PUT-Vorgang für den Stream im angegebenen Zeitraum an. Metrikname auf Shard-Ebene: `IncomingBytes` Abmessungen: StreamName Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben Einheiten: Byte
`IncomingRecords`	Anzahl der Datensätze, die im angegebenen Zeitraum erfolgreich an den Kinesis-Stream übertragen wurden. Diese Metrik enthält die Anzahlen der Datensätze von `PutRecord` - und `PutRecords` -Vorgängen. Die Statistiken Minimum, Maximum und Durchschnitt geben die Datensätze in einem einzelnen PUT-Vorgang für den Stream im angegebenen Zeitraum an. Metrikname auf Shard-Ebene: `IncomingRecords` Abmessungen: StreamName Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben Einheiten: Anzahl

Metrik	Description
PutRecord.Bytes	Anzahl der Bytes, die im angegebenen Zeitraum mithilfe des Vorgangs PutRecord an den Kinesis-Stream übertragen wurden. Abmessungen: StreamName Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben Einheiten: Byte
PutRecord.Latency	Die Dauer pro PutRecord -Vorgang, gemessen im angegebenen Zeitraum. Abmessungen: StreamName Statistiken: Minimum, Maximum, Durchschnitt Einheiten: Millisekunden
PutRecord.Success	Anzahl der erfolgreichen PutRecord -Vorgänge pro Kinesis-Stream im angegebenen Zeitraum. Der Durchschnitt gibt den Prozentsatz der erfolgreichen Schreibvorgänge in einem Stream an. Abmessungen: StreamName Gültige Statistiken: Durchschnitt, Summe, Stichproben Einheiten: Anzahl

Metrik	Description
PutRecords.Bytes	Anzahl der Bytes, die im angegebenen Zeitraum mithilfe des Vorgangs PutRecords an den Kinesis-Stream übertragen wurden. Abmessungen: StreamName Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben Einheiten: Byte
PutRecords.Latency	Die Dauer pro PutRecords -Vorgang, gemessen im angegebenen Zeitraum. Abmessungen: StreamName Statistiken: Minimum, Maximum, Durchschnitt Einheiten: Millisekunden
PutRecords.Records	Diese Metrik ist veraltet. Verwenden Sie PutRecords.SuccessfulRecords . Abmessungen: StreamName Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben Einheiten: Anzahl
PutRecords.Success	Anzahl der PutRecords -Vorgänge pro Kinesis-Stream mit mindestens einem erfolgreichen Datensatz, gemessenen im angegebenen Zeitraum. Abmessungen: StreamName Gültige Statistiken: Durchschnitt, Summe, Stichproben Einheiten: Anzahl

Metrik	Description
PutRecords.TotalRecords	Die gesamte Anzahl der Datensätze in einem PutRecords -Vorgang pro Kinesis-Datenstrom, gemessenen im angegebenen Zeitraum. Abmessungen: StreamName Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben Einheiten: Anzahl
PutRecords.SuccessfulRecords	Anzahl der erfolgreichen Datensätze in einem PutRecords -Vorgang pro Kinesis-Datenstrom, gemessenen im angegebenen Zeitraum. Abmessungen: StreamName Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben Einheiten: Anzahl
PutRecords.FailedRecords	Anzahl der Datensätze, die aufgrund von internen Fehlern in einem PutRecords -Vorgang pro Kinesis-Datenstrom zurückgewiesen wurden, gemessen über den angegebenen Zeitraum. Gelegentliche interne Fehler sind zu erwarten und sollten erneut versucht werden. Abmessungen: StreamName Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben Einheiten: Anzahl

Metrik	Description
PutRecords.ThrottledRecords	Anzahl der Datensätze, die aufgrund von Drosselung in einem PutRecords -Vorgang pro Kinesis-Datenstrom zurückgewiesen wurden, gemessen über den angegebenen Zeitraum. Abmessungen: StreamName Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben Einheiten: Anzahl
ReadProvisionedThroughputExceeded	Anzahl von GetRecords -Aufrufen, die im angegebenen Zeitraum für den Stream gedrosselt wurden. Die am häufigsten verwendete Statistik für diese Metrik ist Durchschnitt. Wenn die Statistik Minimum den Wert 1 hat, wurden im angegebenen Zeitraum sämtliche Datensätze für den Stream gedrosselt. Wenn die Statistik Maximum den Wert 0 (null) hat, wurden im angegebenen Zeitraum keine Datensätze für den Stream gedrosselt. Metrikname auf Shard-Ebene: ReadProvisionedThroughputExceeded Abmessungen: StreamName Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben Einheiten: Anzahl

Metrik	Description
`SubscribeToShard.RateExceeded`	Diese Metrik wird ausgelöst, wenn ein neuer Abonnementversuch fehlschlägt, da bereits ein aktives Abonnement von demselben Verbraucher existiert, oder wenn Sie die Anzahl der Aufrufe pro Sekunde überschreiten, die für diese Operation zulässig ist. Abmessungen: StreamName, ConsumerName
`SubscribeToShard.Success`	Diese Metrik zeichnet auf, ob das SubscribeToShard Abonnement erfolgreich eingerichtet wurde. Das Abonnement besteht nur für maximal 5 Minuten. Daher wird diese Metrik mindestens einmal alle 5 Minuten ausgegeben. Abmessungen: StreamName, ConsumerName
`SubscribeToShardEvent.Bytes`	Anzahl der Bytes, die im angegebenen Zeitraum vom Shard empfangen wurden. Die Statistiken Minimum, Maximum und Durchschnitt geben die Bytes an, die in einem einzelnen Ereignis für den angegebenen Zeitraum veröffentlicht wurden. Metrikname auf Shard-Ebene: `OutgoingBytes` Abmessungen: StreamName, ConsumerName Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben Einheiten: Byte

Metrik	Description
`SubscribeToShardEvent.MillisBehindLatest`	Die Anzahl der Millisekunden, die die gelesenen Datensätze von der Spitze des Streams entfernt sind, gibt an, wie weit der Verbraucher hinter der aktuellen Uhrzeit zurückliegt. Abmessungen:, StreamName ConsumerName Gültige Statistiken: Minimum, Maximum, Durchschnitt, Stichproben Einheiten: Millisekunden
`SubscribeToShardEvent.Records`	Anzahl der Datensätze, die im angegebenen Zeitraum vom Shard empfangen wurden. Die Statistiken Minimum, Maximum und Durchschnitt geben die Datensätze in einem einzelnen Ereignis für den angegebenen Zeitraum an. Metrikname auf Shard-Ebene: `OutgoingRecords` Abmessungen: StreamName, ConsumerName Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben Einheiten: Anzahl
`SubscribeToShardEvent.Success`	Diese Metrik wird jedes Mal ausgegeben, wenn ein Ereignis erfolgreich veröffentlicht wird. Es wird nur ausgegeben, wenn ein aktives Abonnement besteht. Abmessungen: StreamName, ConsumerName Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben Einheiten: Anzahl

Metrik	Description
`WriteProvisionedThroughputExceeded`	Anzahl der Datensätze, die im angegebenen Zeitraum infolge der Drosselung für den Stream abgelehnt wurden. Diese Metrik enthält die Drosselung infolge von `PutRecord` - und `PutRecords` -Vorgängen. Die am häufigsten verwendete Statistik für diese Metrik ist Durchschnitt. Wenn die Statistik Minimum einen Wert ungleich null hat, wurden im angegebenen Zeitraum Datensätze für den Stream gedrosselt. Wenn die Statistik Maximum den Wert 0 (null) hat, wurden im angegebenen Zeitraum keine Datensätze für den Stream gedrosselt. Metrikname auf Shard-Ebene: `WriteProvisionedThroughputExceeded` Abmessungen: StreamName Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben Einheiten: Anzahl

Verbesserte Metriken auf Shard-Ebene

Der `AWS/Kinesis`-Namespace enthält die folgenden Metriken auf Shard-Ebene.

Kinesis sendet die folgenden Shard-Level-Metriken an CloudWatch jede Minute. Jede Metrikdimension erstellt eine CloudWatch Metrik und führt ungefähr 43.200 `PutMetricData` API-Aufrufe pro Monat durch. Diese Metriken sind nicht standardmäßig aktiviert. Von Kinesis übermittelte erweiterte Metriken sind kostenpflichtig. Weitere Informationen finden Sie unter Amazon CloudWatch Pricing unter der Überschrift Amazon CloudWatch Custom Metrics. Die Kosten fallen pro Shard pro Metrik pro Monat an.

Metrik	Description
IncomingBytes	Anzahl der Byte, die im angegebenen Zeitraum erfolgreich an den Shard übertragen wurden. Diese Metrik enthält die Byte von PutRecord - und PutRecords -Vorgängen. Die Statistiken Minimum, Maximum und Durchschnitt geben die Byte in einem einzelnen PUT-Vorgang für den Shard im angegebenen Zeitraum an. Metrikname auf Stream-Ebene: IncomingBytes Abmessungen: StreamName, ShardId Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben Einheiten: Byte
IncomingRecords	Anzahl der Datensätze, die im angegebenen Zeitraum erfolgreich an den Shard übertragen wurden. Diese Metrik enthält die Anzahlen der Datensätze von PutRecord - und PutRecords -Vorgängen. Die Statistiken Minimum, Maximum und Durchschnitt geben die Datensätze in einem einzelnen PUT-Vorgang für den Shard im angegebenen Zeitraum an. Metrikname auf Stream-Ebene: IncomingRecords Abmessungen: StreamName, ShardId Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben Einheiten: Anzahl
IteratorAgeMilliseconds	Alter des letzten Datensatzes in allen GetRecords -Aufrufen an einen Shard, gemessen im angegebenen Zeitraum. Das Alter ist die Differenz zwischen der aktuellen Zeit und der Zeit, zu der der letzte Datensatz

Metrik	Description
	des `GetRecords` -Aufrufs in den Stream geschrieben wurde. Die Statistiken Minimum und Maximum können zum Nachverfolgen des Fortschritts von Kinesis-V erbraucheranwendungen verwendet werden. Wenn der Wert 0 (null) beträgt, sind die gelesenen Datensätze vollständig mit dem Stream aktualisiert. Metrikname auf Stream-Ebene: `GetRecords.IteratorAgeMilliseconds` Abmessungen: StreamName, ShardId Gültige Statistiken: Minimum, Maximum, Durchschnitt, Stichproben Einheiten: Millisekunden
`OutgoingBytes`	Anzahl der Byte, die im angegebenen Zeitraum vom Shard abgerufen wurden. Die Statistiken Minimum, Maximum und Durchschnitt geben die Bytes an, die für den Shard im angegebenen Zeitraum in einem einzelnen `GetRecords` -Vorgang zurückgegeben oder bei einem einzelnen `SubscribeToShard` -Ereignis veröffentlicht wurden. Metrikname auf Stream-Ebene: `GetRecords.Bytes` Abmessungen: StreamName, ShardId Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben Einheiten: Byte

Metrik	Description
`OutgoingRecords`	Anzahl der Datensätze, die im angegebenen Zeitraum vom Shard abgerufen wurden. Die Statistiken Minimum, Maximum und Durchschnitt geben die Datensätze an, die für den Shard im angegebenen Zeitraum in einem einzelnen `GetRecords` -Vorgang zurückgegeben oder bei einem einzelnen `SubscribeToShard` -Ereignis veröffentlicht wurden. Metrikname auf Stream-Ebene: `GetRecords.Records` Abmessungen: StreamName, ShardId Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben Einheiten: Anzahl

Metrik	Description
`ReadProvisionedThroughputExceeded`	Anzahl von `GetRecords` -Aufrufen, die im angegebenen Zeitraum für den Shard gedrosselt wurden. Diese Anzahl von Ausnahmen deckt alle Dimensionen mit folgenden Einschränkungen ab: 5 Lesevorgänge pro Shard pro Sekunde oder 2 MB pro Sekunde pro Shard. Die am häufigsten verwendete Statistik für diese Metrik ist Durchschnitt. Wenn die Statistik Minimum den Wert 1 hat, wurden im angegebenen Zeitraum sämtliche Datensätze für den Shard gedrosselt. Wenn die Statistik Maximum den Wert 0 (null) hat, wurden im angegebenen Zeitraum keine Datensätze für den Shard gedrosselt. Metrikname auf Stream-Ebene: `ReadProvisionedThroughputExceeded` Abmessungen: StreamName, ShardId Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben Einheiten: Anzahl

Metrik	Description
`WriteProvisionedThroughputExceeded`	Anzahl der Datensätze, die im angegebenen Zeitraum infolge der Drosselung für den Shard abgelehnt wurden. Diese Metrik enthält die Drosselung infolge von `PutRecord` - und `PutRecords` -Vorgängen und deckt alle Dimensionen mit folgenden Einschränkungen ab: 1 000 Lesevorgänge pro Shard pro Sekunde oder 1 MB pro Sekunde pro Shard. Die am häufigsten verwendete Statistik für diese Metrik ist Durchschnitt. Wenn die Statistik Minimum einen Wert ungleich null hat, wurden im angegebenen Zeitraum Datensätze für den Shard gedrosselt. Wenn die Statistik Maximum den Wert 0 (null) hat, wurden im angegebenen Zeitraum keine Datensätze für den Shard gedrosselt. Metrikname auf Stream-Ebene: `WriteProvisionedThroughputExceeded` Abmessungen: StreamName, ShardId Gültige Statistiken: Minimum, Maximum, Durchschnitt, Summe, Stichproben Einheiten: Anzahl

Dimensionen für Amazon Kinesis Data Streams Streams-Metriken

Dimension	Description
`StreamName`	Der Name des Kinesis-Streams. Alle verfügbaren Statistiken werden von `StreamName` gefiltert.

Empfohlene Amazon Kinesis Data Streams Streams-Metriken

Verschiedene Metriken für Amazon Kinesis Data Streams könnten für Kunden von Kinesis Data Streams von besonderem Interesse sein. Die folgende Liste enthält empfohlene Metriken und ihre Verwendung.

Metrik	Nutzungshinweise
`GetRecords.IteratorAgeMilliseconds`	Verfolgt die Leseposition über alle Shards und Kunden in dem Stream. Wenn das Alter eines Iterators 50 % des Aufbewahrungszeitraums (standardmäßig 24 Stunden, anpassbar auf bis zu 7 Tage) überschreitet, besteht die Gefahr des Datenverlusts durch Ablauf des Datensatzes. Wir empfehlen Ihnen, CloudWatch Alarme für die Maximum-Statistik zu verwenden, um Sie zu warnen, bevor dieser Verlust ein Risiko darstellt. Ein Beispielszenario mit dieser Metrik finden Sie unter Die Verarbeitung von Verbraucherdaten hinkt hinterher.
`ReadProvisionedThroughputExceeded`	Wenn Ihre konsumentenseitige Datensatzverarbeitung zurückfällt, ist es oft schwierig zu erkennen, wo der Engpass ist. Verwenden Sie diese Metrik, um zu bestimmen, ob Ihre Lesevorgänge aufgrund einer Überschreitung Ihrer Durchsatzgrenzwerte gedrosselt werden. Die am häufigsten verwendete Statistik für diese Metrik ist Durchschnitt.
`WriteProvisionedThroughputExceeded`	Diese dient dem gleichen Zweck wie die Metrik `ReadProvisionedThroughputExceeded` , jedoch für die Producer- (Put) Seite des Streams. Die am häufigsten verwendete Statistik für diese Metrik ist Durchschnitt.
`PutRecord.Success` , `PutRecords.Success`	Wir empfehlen die Verwendung von CloudWatch Alarmen in der Statistik „Durchschnitt“, um anzuzeigen, wenn Datensätze nicht in den Stream aufgenommen werden. Wählen Sie einen oder beide Put-Typen, je nachdem, was Ihr Producer verwendet. Wenn Sie die Amazon Kinesis Producer Library (KPL) verwenden, verwenden Sie. `PutRecords.Success`
`GetRecords.Success`	Wir empfehlen die Verwendung von CloudWatch Alarmen in der Durchschnittsstatistik, um anzuzeigen, wenn Datensätze aus dem Stream fehlerhaft sind.

Greifen Sie auf CloudWatch Amazon-Metriken für Kinesis Data Streams zu

Sie können Metriken für Kinesis Data Streams über die CloudWatch Konsole, die Befehlszeile oder die CloudWatch API überwachen. Die folgenden Verfahren zeigen, wie Sie mithilfe dieser verschiedenen Verfahren auf die Metriken zugreifen können.

So greifen Sie über die Konsole auf Metriken zu CloudWatch

1. Öffnen Sie die CloudWatch Konsole unter https://console.aws.amazon.com/cloudwatch/.
2. Wählen Sie in der Navigationsleiste eine Region aus.
3. Wählen Sie im Navigationsbereich Metriken aus.
4. Wählen Sie im Bereich CloudWatch Metriken nach Kategorie die Option Kinesis Metrics aus.
5. Klicken Sie auf die entsprechende Zeile, um die Statistiken für das angegebene MetricNameund StreamNameanzuzeigen.

   Hinweis: Die meisten Namen der Konsolenstatistiken stimmen mit den entsprechenden CloudWatch Metriknamen überein, die zuvor aufgeführt wurden, mit Ausnahme von Lesedurchsatz und Schreibdurchsatz. Diese Statistiken werden in Intervallen von 5 Minuten berechnet: Der Schreibdurchsatz überwacht die `IncomingBytes` CloudWatch Metrik und der Lesedurchsatz überwacht `GetRecords.Bytes` die Metrik.
6. (Optional) Wählen Sie im Diagrammbereich eine Statistik und einen Zeitraum aus, und erstellen Sie dann mit diesen Einstellungen einen CloudWatch Alarm.

Um auf Metriken zuzugreifen, verwenden Sie AWS CLI

Verwenden Sie die Listen-Metriken und get-metric-statisticsBefehle.

So greifen Sie mit der CloudWatch CLI auf Metriken zu

Verwenden Sie die Befehle mon-list-metrics und mon-get-stats.

Um über die CloudWatch API auf Metriken zuzugreifen

Verwenden Sie die Operationen ListMetrics und GetMetricStatistics.

Überwachen Sie den Zustand des Kinesis Data Streams Streams-Agenten mit Amazon CloudWatch

Der Agent veröffentlicht benutzerdefinierte CloudWatch Metriken mit dem Namespace. AWS KinesisAgent Diese Metriken ermöglichen die Einschätzung, ob der Agent Daten wie angegeben an Kinesis Data Streams sendet, ob er in gutem Zustand ist und angemessene CPU- und Arbeitsspeicherressourcen auf dem Datenproduzenten konsumiert. Metriken wie die Anzahl der gesendeten Datensätze und Byte sind nützlich, um die Geschwindigkeit zu verstehen, mit der der Agent Daten an den Stream sendet. Wenn diese Metriken um einige Prozent unter die erwarteten Schwellenwerte oder auf Null sinken, kann dies auf Probleme mit der Konfiguration, Netzwerkfehler oder Probleme mit dem Zustand des Agenten hinweisen. Metriken wie On-Host CPU- und Speicherbelegung sowie die Anzahl der Agentenfehler zeigen die Nutzung der Ressourcen des Datenproduzenten an und informieren über potenzielle Konfigurations- oder Hostfehler. Schließlich protokolliert der Agent auf Serviceausnahmen, um die Untersuchung von Agentenproblemen zu unterstützen. Diese Metriken werden in der Region gemeldet, die in der Agentenkonfiguration angegeben ist`cloudwatch.endpoint`. CloudWatch Metriken, die von mehreren Kinesis-Agenten veröffentlicht wurden, werden aggregiert oder kombiniert. Weitere Informationen zur Agentenkonfiguration finden Sie unter Geben Sie die Einstellungen für die Agentenkonfiguration an

Überwachen Sie mit CloudWatch

Der Kinesis Data Streams Streams-Agent sendet die folgenden Metriken an CloudWatch.

Metrik	Description
`BytesSent`	Die Anzahl von Byte, die über den angegebenen Zeitraum an Kinesis Data Streams gesendet wurden. Einheiten: Byte
`RecordSendAttempts`	Die Anzahl der versuchten Datensätze (zum ersten Mal oder wiederholt) in einem Aufruf an `PutRecords` in dem angegebenen Zeitraum. Einheiten: Anzahl
`RecordSendErrors`	Die Anzahl der Datensätze mit Fehlerstatus in einem Aufruf an `PutRecords` , einschließlich wiederholter Versuche, in dem angegebenen Zeitraum.

Metrik	Description
	Einheiten: Anzahl
`ServiceErrors`	Die Anzahl der Aufrufe an `PutRecords` , die zu einem Servicefehler führten (außer Ablehnungsfehlern) in dem angegebenen Zeitraum. Einheiten: Anzahl

API-Aufrufe von Amazon Kinesis Data Streams protokollieren mit AWS CloudTrail

Amazon Kinesis Data Streams ist in einen Service integriert AWS CloudTrail, der eine Aufzeichnung der Aktionen bereitstellt, die von einem Benutzer, einer Rolle oder einem AWS Service in Kinesis Data Streams ausgeführt wurden. CloudTrail erfasst alle API-Aufrufe für Kinesis Data Streams als Ereignisse. Zu den erfassten Aufrufen gehören Aufrufe von der Kinesis-Data-Streams-Konsole und Code-Aufrufe der API-Operationen von Kinesis Data Streams. Wenn Sie einen Trail erstellen, können Sie die kontinuierliche Bereitstellung von CloudTrail Ereignissen an einen Amazon S3 S3-Bucket aktivieren, einschließlich Ereignissen für Kinesis Data Streams. Wenn Sie keinen Trail konfigurieren, können Sie die neuesten Ereignisse trotzdem in der CloudTrail Konsole im Ereignisverlauf anzeigen. Anhand der von gesammelten Informationen können Sie die Anfrage CloudTrail, die an Kinesis Data Streams gestellt wurde, die IP-Adresse, von der aus die Anfrage gestellt wurde, wer die Anfrage gestellt hat, wann sie gestellt wurde, und weitere Details ermitteln.

Weitere Informationen darüber CloudTrail, einschließlich der Konfiguration und Aktivierung, finden Sie im AWS CloudTrail Benutzerhandbuch.

Kinesis Data Streams Streams-Informationen in CloudTrail

CloudTrail ist für Ihr AWS Konto aktiviert, wenn Sie das Konto erstellen. Wenn unterstützte Ereignisaktivitäten in Kinesis Data Streams auftreten, wird diese Aktivität zusammen mit anderen AWS Serviceereignissen im CloudTrail Ereignisverlauf in einem Ereignis aufgezeichnet. Sie können aktuelle Ereignisse in Ihrem AWS Konto anzeigen, suchen und herunterladen. Weitere Informationen finden Sie unter Anzeigen von Ereignissen mit dem CloudTrail -API-Ereignisverlauf.

Für eine fortlaufende Aufzeichnung von Ereignissen in Ihrem AWS Konto, einschließlich Ereignissen für Kinesis Data Streams, erstellen Sie einen Trail. Ein Trail ermöglicht CloudTrail die Übermittlung von Protokolldateien an einen Amazon S3 S3-Bucket. Wenn Sie einen Trail in der Konsole erstellen,

gilt der Trail standardmäßig für alle AWS Regionen. Der Trail protokolliert Ereignisse aus allen Regionen der AWS Partition und übermittelt die Protokolldateien an den von Ihnen angegebenen Amazon S3 S3-Bucket. Darüber hinaus können Sie andere AWS Dienste konfigurieren, um die in den CloudTrail Protokollen gesammelten Ereignisdaten weiter zu analysieren und darauf zu reagieren. Weitere Informationen finden Sie hier:

- Übersicht zum Erstellen eines Trails
- CloudTrail Unterstützte Dienste und Integrationen
- Konfiguration von Amazon SNS SNS-Benachrichtigungen für CloudTrail
- Empfangen von CloudTrail Protokolldateien aus mehreren Regionen und Empfangen von CloudTrail Protokolldateien von mehreren Konten

Kinesis Data Streams unterstützt die Protokollierung der folgenden Aktionen als Ereignisse in CloudTrail Protokolldateien:

- AddTagsToStream
- CreateStream
- DecreaseStreamRetentionPeriod
- DeleteStream
- DeregisterStreamConsumer
- DescribeStream
- DescribeStreamConsumer
- DisableEnhancedMonitoring
- EnableEnhancedMonitoring
- GetRecords
- GetShardIterator
- IncreaseStreamRetentionPeriod
- ListStreamConsumers
- ListStreams
- ListTagsForStream
- MergeShards
- PutRecord
- PutRecords

- RegisterStreamConsumer
- RemoveTagsFromStream
- SplitShard
- StartStreamEncryption
- StopStreamEncryption
- SubscribeToShard
- UpdateShardCount
- UpdateStreamMode

Jeder Ereignis- oder Protokolleintrag enthält Informationen zu dem Benutzer, der die Anforderung generiert hat. Die Identitätsinformationen unterstützen Sie bei der Ermittlung der folgenden Punkte:

- Ob die Anfrage mit Root- oder AWS Identity and Access Management (IAM-) Benutzeranmeldedaten gestellt wurde.
- Gibt an, ob die Anforderung mit temporären Sicherheitsanmeldeinformationen für eine Rolle oder einen Verbundbenutzer gesendet wurde.
- Ob die Anfrage von einem anderen AWS Dienst gestellt wurde.

Weitere Informationen finden Sie unter CloudTrail userIdentity-Element.

Beispiel: Einträge in der Kinesis Data Streams Streams-Protokolldatei

Ein Trail ist eine Konfiguration, die die Übertragung von Ereignissen als Protokolldateien an einen von Ihnen angegebenen Amazon S3 S3-Bucket ermöglicht. CloudTrail Protokolldateien enthalten einen oder mehrere Protokolleinträge. Ein Ereignis ist eine einzelne Anforderung aus einer beliebigen Quelle und enthält Informationen zur angeforderten Aktion, zu Datum und Uhrzeit der Aktion, zu den Anforderungsparametern usw. CloudTrail -Protokolldateien stellen kein geordnetes Stack-Trace der öffentlichen API-Aufrufe dar. Daher werden sie nicht in einer bestimmten Reihenfolge angezeigt.

Das folgende Beispiel zeigt einen CloudTrail Protokolleintrag, der die `MergeShards` Aktionen`CreateStream`,`DescribeStream`,`ListStreams`, `DeleteStreamSplitShard`, und demonstriert.

```
{
    "Records": [
        {
```

```
            "eventVersion": "1.01",
            "userIdentity": {
                "type": "IAMUser",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::012345678910:user/Alice",
                "accountId": "012345678910",
                "accessKeyId": "EXAMPLE_KEY_ID",
                "userName": "Alice"
            },
            "eventTime": "2014-04-19T00:16:31Z",
            "eventSource": "kinesis.amazonaws.com",
            "eventName": "CreateStream",
            "awsRegion": "us-east-1",
            "sourceIPAddress": "127.0.0.1",
            "userAgent": "aws-sdk-java/unknown-version Linux/x.xx",
            "requestParameters": {
                "shardCount": 1,
                "streamName": "GoodStream"
            },
            "responseElements": null,
            "requestID": "db6c59f8-c757-11e3-bc3b-57923b443c1c",
            "eventID": "b7acfcd0-6ca9-4ee1-a3d7-c4e8d420d99b"
        },
        {
            "eventVersion": "1.01",
            "userIdentity": {
                "type": "IAMUser",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::012345678910:user/Alice",
                "accountId": "012345678910",
                "accessKeyId": "EXAMPLE_KEY_ID",
                "userName": "Alice"
            },
            "eventTime": "2014-04-19T00:17:06Z",
            "eventSource": "kinesis.amazonaws.com",
            "eventName": "DescribeStream",
            "awsRegion": "us-east-1",
            "sourceIPAddress": "127.0.0.1",
            "userAgent": "aws-sdk-java/unknown-version Linux/x.xx",
            "requestParameters": {
                "streamName": "GoodStream"
            },
            "responseElements": null,
            "requestID": "f0944d86-c757-11e3-b4ae-25654b1d3136",
```

```
            "eventID": "0b2f1396-88af-4561-b16f-398f8eaea596"
        },
        {
            "eventVersion": "1.01",
            "userIdentity": {
                "type": "IAMUser",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::012345678910:user/Alice",
                "accountId": "012345678910",
                "accessKeyId": "EXAMPLE_KEY_ID",
                "userName": "Alice"
            },
            "eventTime": "2014-04-19T00:15:02Z",
            "eventSource": "kinesis.amazonaws.com",
            "eventName": "ListStreams",
            "awsRegion": "us-east-1",
            "sourceIPAddress": "127.0.0.1",
            "userAgent": "aws-sdk-java/unknown-version Linux/x.xx",
            "requestParameters": {
                "limit": 10
            },
            "responseElements": null,
            "requestID": "a68541ca-c757-11e3-901b-cbcfe5b3677a",
            "eventID": "22a5fb8f-4e61-4bee-a8ad-3b72046b4c4d"
        },
        {
            "eventVersion": "1.01",
            "userIdentity": {
                "type": "IAMUser",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::012345678910:user/Alice",
                "accountId": "012345678910",
                "accessKeyId": "EXAMPLE_KEY_ID",
                "userName": "Alice"
            },
            "eventTime": "2014-04-19T00:17:07Z",
            "eventSource": "kinesis.amazonaws.com",
            "eventName": "DeleteStream",
            "awsRegion": "us-east-1",
            "sourceIPAddress": "127.0.0.1",
            "userAgent": "aws-sdk-java/unknown-version Linux/x.xx",
            "requestParameters": {
                "streamName": "GoodStream"
            },
```

```
            "responseElements": null,
            "requestID": "f10cd97c-c757-11e3-901b-cbcfe5b3677a",
            "eventID": "607e7217-311a-4a08-a904-ec02944596dd"
        },
        {
            "eventVersion": "1.01",
            "userIdentity": {
                "type": "IAMUser",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::012345678910:user/Alice",
                "accountId": "012345678910",
                "accessKeyId": "EXAMPLE_KEY_ID",
                "userName": "Alice"
            },
            "eventTime": "2014-04-19T00:15:03Z",
            "eventSource": "kinesis.amazonaws.com",
            "eventName": "SplitShard",
            "awsRegion": "us-east-1",
            "sourceIPAddress": "127.0.0.1",
            "userAgent": "aws-sdk-java/unknown-version Linux/x.xx",
            "requestParameters": {
                "shardToSplit": "shardId-000000000000",
                "streamName": "GoodStream",
                "newStartingHashKey": "11111111"
            },
            "responseElements": null,
            "requestID": "a6e6e9cd-c757-11e3-901b-cbcfe5b3677a",
            "eventID": "dcd2126f-c8d2-4186-b32a-192dd48d7e33"
        },
        {
            "eventVersion": "1.01",
            "userIdentity": {
                "type": "IAMUser",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::012345678910:user/Alice",
                "accountId": "012345678910",
                "accessKeyId": "EXAMPLE_KEY_ID",
                "userName": "Alice"
            },
            "eventTime": "2014-04-19T00:16:56Z",
            "eventSource": "kinesis.amazonaws.com",
            "eventName": "MergeShards",
            "awsRegion": "us-east-1",
            "sourceIPAddress": "127.0.0.1",
```

```
            "userAgent": "aws-sdk-java/unknown-version Linux/x.xx",
            "requestParameters": {
                "streamName": "GoodStream",
                "adjacentShardToMerge": "shardId-000000000002",
                "shardToMerge": "shardId-000000000001"
            },
            "responseElements": null,
            "requestID": "e9f9c8eb-c757-11e3-bf1d-6948db3cd570",
            "eventID": "77cf0d06-ce90-42da-9576-71986fec411f"
        }
    ]
}
```

Überwachen Sie die Kinesis-Clientbibliothek mit Amazon CloudWatch

Die Kinesis Client Library (KCL) für Amazon Kinesis Data Streams veröffentlicht in Ihrem Namen benutzerdefinierte CloudWatch Amazon-Metriken, wobei der Name Ihrer KCL-Anwendung als Namespace verwendet wird. Sie können diese Metriken anzeigen, indem Sie zur Konsole navigieren und Benutzerdefinierte Metriken auswählen. CloudWatch Weitere Informationen zu benutzerdefinierten Metriken finden Sie unter Veröffentlichen benutzerdefinierter Metriken im CloudWatch Amazon-Benutzerhandbuch.

Für die CloudWatch von der KCL hochgeladenen Metriken wird eine geringe Gebühr erhoben. Insbesondere fallen Gebühren für Amazon CloudWatch Custom Metrics und Amazon CloudWatch API Requests an. Weitere Informationen finden Sie unter CloudWatch Amazon-Preise.

Themen

- Metriken und Namespace
- Metrische Ebenen und Dimensionen
- Konfiguration der Metrik
- Liste der Metriken

Metriken und Namespace

Der zum Hochladen von Kennzahlen verwendete Namespace ist der Anwendungsname, den Sie beim Starten von KCL angeben.

Metrische Ebenen und Dimensionen

Es gibt zwei Möglichkeiten, um zu steuern, welche Metriken zu CloudWatch hochgeladen werden:

Metrikstufen

Jeder Metrik ist eine individuelle Stufe zugewiesen. Wenn Sie eine Berichtsebene für Kennzahlen festlegen, werden Kennzahlen, deren individuelle Ebene unter der Berichtsebene liegt, nicht an diese gesendet CloudWatch. Die Stufen sind: `NONE`, `SUMMARY` und `DETAILED`. Die Standardeinstellung ist`DETAILED`; das heißt, alle Metriken werden an gesendet CloudWatch. Die Berichtsstufe `NONE` bedeutet, dass keine Kennzahlen gesendet werden. Informationen dazu, welchen Metriken welche Stufen zugewiesen werden, finden Sie unter Liste der Metriken.

aktivierte Dimensionen

Jeder KCL-Metrik sind Dimensionen zugeordnet, an die auch gesendet CloudWatch werden. Wenn KCL in KCL 2.x für die Verarbeitung eines einzelnen Datenstroms konfiguriert ist, sind alle Metrikdimensionen (`Operation`, `ShardId` und `WorkerIdentifier`) standardmäßig aktiviert. Außerdem kann in KCL 2.x die Dimension `Operation` nicht deaktiviert werden, wenn KCL für die Verarbeitung eines einzelnen Datenstroms konfiguriert ist. Wenn KCL in KCL 2.x für die Verarbeitung mehrerer Datenströme konfiguriert ist, sind alle Metrikdimensionen (`Operation`, `ShardIdStreamId`, und`WorkerIdentifier`) standardmäßig aktiviert. Wenn KCL in KCL 2.x für die Verarbeitung mehrerer Datenströme konfiguriert ist, können außerdem die Dimensionen `Operation` und die Dimensionen nicht deaktiviert werden. `StreamId` `StreamId`Die Dimension ist nur für die Metriken pro Shard verfügbar.

In KCL 1.x sind nur die Dimensionen `Operation` und `ShardId` standardmäßig aktiviert, die Dimension `WorkerIdentifier` ist deaktiviert. In KCL 1.x kann die `Operation`-Dimension nicht deaktiviert werden.

Weitere Informationen zu CloudWatch metrischen Dimensionen finden Sie im Abschnitt Dimensionen des Themas Amazon CloudWatch Concepts im CloudWatch Amazon-Benutzerhandbuch.

Wenn die `WorkerIdentifier` Dimension aktiviert ist und bei jedem Neustart eines bestimmten KCL-Workers ein anderer Wert für die Worker-ID-Eigenschaft verwendet wird, werden neue Messwerte mit neuen `WorkerIdentifier` Dimensionswerten an gesendet. CloudWatch Wenn der `WorkerIdentifier`-Dimensionswert für verschiedene KCL-Worker-Neustarts identisch sein muss, müssen Sie explizit denselben Worker-ID-Wert bei der Initialisierung für jeden Worker

angeben. Beachten Sie, dass der Worker-ID-Wert für jeden aktiven KCL-Worker unter allen KCL-Workern eindeutig sein muss.

Konfiguration der Metrik

Metrikebenen und aktivierte Dimensionen können mithilfe der KinesisClientLibConfiguration Instanz konfiguriert werden, die beim Starten der KCL-Anwendung an Worker übergeben wird. In MultiLangDaemon diesem Fall können die `metricsEnabledDimensions` Eigenschaften `metricsLevel` und in der Datei .properties angegeben werden, die zum Starten der MultiLangDaemon KCL-Anwendung verwendet wird.

Metrikstufen kann einer von drei Werten zugewiesen werden: NONE, SUMMARY oder DETAILED. Bei den Werten für aktivierte Dimensionen muss es sich um durch Kommas getrennte Zeichenketten mit der Liste der Dimensionen handeln, die für die Metriken zulässig sind. CloudWatch Die von der KCL-Anwendung verwendeten Dimensionen sind `Operation`, `ShardId` und `WorkerIdentifier`.

Liste der Metriken

Die folgenden Tabellen listen die KCL-Metriken, gruppiert nach Umfang und Operation, auf.

Themen

- Per-KCL-application Metriken
- Kennzahlen pro Mitarbeiter
- Metriken pro Shard

Per-KCL-application Metriken

Diese Metriken werden für alle KCL-Worker innerhalb des Anwendungsbereichs aggregiert, wie im CloudWatch Amazon-Namespace definiert.

Themen

- LeaseAssignmentManager
- InitializeTask
- ShutdownTask
- ShardSyncTask
- BlockOnParentTask

- PeriodicShardSyncManager
- MultistreamTracker

LeaseAssignmentManager

Der `LeaseAssignmentManager` Betrieb ist dafür verantwortlich, den Mitarbeitern Leasingverträge zuzuweisen und die Mietverträge zwischen den Mitarbeitern neu auszurichten, um eine gleichmäßige Auslastung der Mitarbeiterressourcen zu erreichen. Die Logik für diesen Vorgang umfasst das Lesen der Leasing-bezogenen Metadaten aus der Leasing-Tabelle und der Kennzahlen aus der Tabelle mit den Arbeitskräfte-Metriken sowie das Durchführen von Leasing-Zuweisungen.

Metrik	Description
LeaseAndWorkerMetricsLoad.Zeit	Zeitaufwand für das Laden aller Leasingverträge und Worker-Metrik-Einträge im Lease Assignment Manager (LAM), dem neuen Algorithmus für Leasingzuweisung und Lastenausgleich, der in KCL 3.x eingeführt wurde. Metrikstufe: Detailed Einheiten: Millisekunden
TotalLeases	Gesamtzahl der Leases für die aktuelle KCL-Anwendung. Metrikstufe: Summary Einheiten: Anzahl
NumWorkers	Gesamtzahl der Mitarbeiter in der aktuellen KCL-Anwendung. Metrikstufe: Summary Einheiten: Anzahl
AssignExpiredOrUnassignedLeases. Zeit	Zeit für die In-Memory-Zuweisung abgelaufener Leases. Metrikstufe: Detailed Einheiten: Millisekunden

Metrik	Description
LeaseSpillover	Anzahl der Leases, die nicht zugewiesen wurden, weil das Limit für die maximale Anzahl von Leases oder den maximalen Durchsatz pro Mitarbeiter erreicht wurde. Metrikstufe: Summary Einheiten: Anzahl
BalanceWorkerVariance. Zeit	Zeit für den speicherinternen Ausgleich von Mietverträgen zwischen Mitarbeitern. Metrikstufe: Detailed Einheiten: Millisekunden
NumOfLeasesReassignment	Gesamtzahl der Neuzuweisungen von Leasingverträgen, die in der aktuellen Iteration der Neuzuweisung vorgenommen wurden. Metrikstufe: Summary Einheiten: Anzahl
FailedAssignmentCount	Anzahl der Fehler bei AssignLease Aufrufen der DynamoDB-Leasetabelle. Metrikstufe: Detailed Einheiten: Anzahl
ParallelyAssignLeases. Zeit	Zeit, neue Zuweisungen in die DynamoDB-Leasetabelle zu übernehmen. Metrikstufe: Detailed Einheiten: Millisekunden
ParallelyAssignLeases. Erfolg	Anzahl der erfolgreich abgewickelten Neuaufträge Metrikstufe: Detailed Einheiten: Anzahl

Metrik	Description
TotalStaleWorkerMetricsEntry	Gesamtzahl der Einträge mit Arbeitskennzahlen, die bereinigt werden müssen. Metrikstufe: Detailed Einheiten: Anzahl
StaleWorkerMetricsCleanup. Zeit	Zeit für das Löschen von Worker-Metrik-Einträgen aus der DynamoDB-Worker-Metriktabelle. Metrikstufe: Detailed Einheiten: Millisekunden
Zeit	Zeit, die der Vorgang in Anspruch genommen hat. `LeaseAssignmentManager` Metrikstufe: Summary Einheiten: Millisekunden
Herzlichen Glückwunsch	Häufigkeit des erfolgreichen Abschlusses der `LeaseAssignmentManager` -Operation. Metrikstufe: Summary Einheiten: Anzahl
ForceLeaderRelease	Zeigt an, dass der Leasingzuweisungsmanager dreimal hintereinander ausgefallen ist und der leitende Mitarbeiter die Führung freigibt. Metrikstufe: Summary Einheiten: Anzahl

Metrik	Description
NumWorkersWithInvalidEntry	Anzahl der Einträge für Arbeitskräftekennzahlen, die als ungültig angesehen werden. Metrikstufe: Summary Einheiten: Anzahl
NumWorkersWithFailingWorkerMetric	Anzahl der Einträge für Arbeitskräftekennzahlen, bei denen -1 (was bedeutet, dass ein Wert für die Arbeitskräftekennzahl nicht verfügbar ist) als einer der Werte für Arbeitskräftekennzahlen angegeben ist. Metrikstufe: Summary Einheiten: Anzahl
LeaseDeserializationFailureCount	Leasingeintrag aus der Leasingtabelle, der nicht deserialisiert werden konnte. Metrikstufe: Summary Einheiten: Anzahl

InitializeTask

Die Operation `InitializeTask` ist verantwortlich für die Initialisierung des Datensatzprozessors für die KCL-Anwendung. Zur Logik dieser Operation gehört der Abruf eines Shard-Iterators von Kinesis Data Streams und die Initialisierung des Datensatzprozessors.

Metrik	Description
KinesisDataFetcher.getIterator.Success	Anzahl der erfolgreichen `GetShardIterator` -Operationen pro KCL-Anwendung. Metrikstufe: Detailed Einheiten: Anzahl

Metrik	Description
KinesisDataFetcher .getIterator.Time	Zeitbedarf für die `GetShardIterator` -Operation für die jeweilige KCL-Anwendung. Metrikstufe: Detailed Einheiten: Millisekunden
RecordProcessor.in itialisieren.TIME	Zeitbedarf für die Initialisierungsmethode des Datensatzprozessors. Metrikstufe: Summary Einheiten: Millisekunden
Herzlichen Glückwuns ch	Anzahl der erfolgreichen Initialisierungen des Datensatzprozessors. Metrikstufe: Summary Einheiten: Anzahl
Zeit	Zeitbedarf des KCL-Workers für die Initialisierung des Datensatz prozessors. Metrikstufe: Summary Einheiten: Millisekunden

ShutdownTask

Die - `ShutdownTask` Operation initiiert die Abschaltsequenz für die Shard-Verarbeitung. Dies kann auftreten, da ein Shard geteilt oder zusammengeführt ist, oder wenn das Shard-Lease vom Worker verloren ging. In beiden Fällen wird die Datensatzprozessorfunktion `shutdown()` aufgerufen. Neue Shards werden auch erkannt, wenn ein Shard geteilt oder zusammengeführt wurde, was zur Erstellung von einem oder zwei neuen Shards führt.

Metrik	Description
CreateLease.Erfolg	Häufigkeit, mit der neue untergeordnete Shards nach dem Schließen des übergeordneten Shards erfolgreich der DynamoDB-Tabelle der KCL-Anwendung hinzugefügt wurden. Metrikstufe: Detailed Einheiten: Anzahl
CreateLease.Zeit	Zeitbedarf für das Hinzufügen von Informationen zu neuen untergeordneten Shards in der DynamoDB-Tabelle der KCL-Anwendung. Metrikstufe: Detailed Einheiten: Millisekunden
UpdateLease.Erfolg	Anzahl der erfolgreichen endgültigen Checkpoints während des Schließens des Datensatzprozessors. Metrikstufe: Detailed Einheiten: Anzahl
UpdateLease.Zeit	Zeitbedarf für die Checkpoint-Operation während des Schließens des Datensatzprozessors. Metrikstufe: Detailed Einheiten: Millisekunden
RecordProcessor.Herunterfahren.Zeit	Zeitbedarf für die Schließungsmethode des Datensatzprozessors. Metrikstufe: Summary Einheiten: Millisekunden
Herzlichen Glückwunsch	Anzahl der erfolgreichen Schließungsvorgänge. Metrikstufe: Summary

Metrik	Description
	Einheiten: Anzahl
Zeit	Zeitbedarf des KCL-Workers für den Schließungsvorgang. Metrikstufe: Summary Einheiten: Millisekunden

ShardSyncTask

Die Operation `ShardSyncTask` erkennt Änderungen an Shard-Informationen für den Kinesis-Datenstrom, damit die KCL-Anwendung neue Shards verarbeiten kann.

Metrik	Description
CreateLease. Erfolg	Anzahl der erfolgreichen Versuche zum Hinzufügen neuer Shard-Inf ormationen in der DynamoDB-Tabelle der KCL-Anwendung. Metrikstufe: Detailed Einheiten: Anzahl
CreateLease.Zeit	Zeitbedarf für das Hinzufügen von Informationen zu neuen Shards in der DynamoDB-Tabelle der KCL-Anwendung. Metrikstufe: Detailed Einheiten: Millisekunden
Herzlichen Glückwuns ch	Anzahl der erfolgreichen Shard Synchronisierungsoperationen. Metrikstufe: Summary Einheiten: Anzahl
Zeit	Zeitbedarf für die Shard-Synchronisierungsoperation. Metrikstufe: Summary

Metrik	Description
	Einheiten: Millisekunden

BlockOnParentTask

Wenn der Shard geteilt oder mit anderen Shards zusammengeführt wird, werden neue untergeordnete Shards erstellt. Die Operation `BlockOnParentTask` stellt sicher, dass die Datensatzverarbeitung für die neuen Shards erst beginnt, wenn die übergeordneten Shards von dem KCL vollständig verarbeitet wurden.

Metrik	Description
Herzlichen Glückwunsch	Anzahl der erfolgreichen Prüfungen für den Abschluss der übergeordneten Shards. Metrikstufe: Summary Einheiten: Anzahl
Zeit	Zeitbedarf für den Abschluss der übergeordneten Shards. Metrikstufe: Summary Einheit: Millisekunden

PeriodicShardSyncManager

`PeriodicShardSyncManager` hat die Aufgabe, die von der KCL-Konsumentenanwendung verarbeiteten Datenströme zu prüfen, Datenströme mit Teil-Leases zu identifizieren und sie zur Synchronisierung weiterzuleiten.

Die folgenden Metriken sind verfügbar, wenn KCL für die Verarbeitung eines einzelnen Datenstroms konfiguriert NumStreamsWithPartialLeases ist (dann wird der Wert von NumStreamsToSync und auf 1 gesetzt) und wenn KCL für die Verarbeitung mehrerer Datenströme konfiguriert ist.

Metrik	Description
NumStreamsToSync	Die Anzahl der Datenströme (pro AWS Konto), die von der Verbrauch eranwendung verarbeitet werden, die Teil-Leases enthält und die zur Synchronisation übergeben werden müssen. Metrikstufe: Summary Einheiten: Anzahl
NumStreamsWithPartialLeases	Die Anzahl der Datenströme (pro AWS Konto), die die Verbrauch eranwendung verarbeitet, die Teilleasingverträge enthält. Metrikstufe: Summary Einheiten: Anzahl
Herzlichen Glückwunsch	Die Anzahl der Male, in denen `PeriodicShardSyncManager` erfolgreich Teil-Leases in den Datenströmen identifizieren konnte, die die Verbraucheranwendung verarbeitet. Metrikstufe: Summary Einheiten: Anzahl
Zeit	Die Zeit (in Millisekunden), die `PeriodicShardSyncManager` benötigt wird, um die Datenströme zu untersuchen, die die Verbrauch eranwendung verarbeitet, um festzustellen, welche Datenströme eine Shard-Synchronisierung erfordern. Metrikstufe: Summary Einheiten: Millisekunden

MultistreamTracker

Die `MultistreamTracker`-Schnittstelle ermöglicht es Ihnen, KCL-Konsumentenanwendungen zu erstellen, die mehrere Datenströme gleichzeitig verarbeiten können.

Metrik	Description
DeletedStreams. Anzahl	Die Anzahl der in diesem Zeitraum gelöschten Datenströme. Metrikstufe: Summary Einheiten: Anzahl
ActiveStreams. Zählen	Die Anzahl der aktiven Datenströme, die verarbeitet werden. Metrikstufe: Summary Einheiten: Anzahl
StreamsPendingDeletion. Zählen	Die Anzahl der Datenströme, die aufgrund von `FormerStreamsLeasesDeletionStrategy` zur Löschung anstehen. Metrikstufe: Summary Einheiten: Anzahl

Kennzahlen pro Mitarbeiter

Diese Metriken werden über alle Datensatzprozessoren, die Daten von einem Kinesis-Datenstrom verbrauchen, etwa einer Amazon-EC2-Instance, aggregiert.

Themen

- WorkerMetricStatsReporter
- LeaseDiscovery
- RenewAllLeases
- TakeLeases

WorkerMetricStatsReporter

Der `WorkerMetricStatReporter` Vorgang ist dafür verantwortlich, in regelmäßigen Abständen Kennzahlen des aktuellen Mitarbeiters in der Tabelle mit den Arbeitskräftekennzahlen zu veröffentlichen. Diese Kennzahlen werden vom `LeaseAssignmentManager` Vorgang zur Ausführung von Leasingaufträgen verwendet.

Metrik	Description
InMemoryMetricStatsReporterFailure	Anzahl der Fehler bei der Erfassung des In-Memory-Worker-Metrikwerts, die auf einen Ausfall einiger Worker-Metriken zurückzuführen sind. Metrikstufe: Summary Einheiten: Anzahl
WorkerMetricStatsReporter. Zeit	Zeit, die die `WorkerMetricsStats` Operation benötigt hat. Metrikstufe: Summary Einheiten: Millisekunden
WorkerMetricStatsReporter. Erfolg	Häufigkeit des erfolgreichen Abschlusses der `WorkerMetricsStats` -Operation. Metrikstufe: Summary Einheiten: Anzahl

LeaseDiscovery

Der `LeaseDiscovery` Vorgang ist dafür verantwortlich, die neuen Leasingverträge zu ermitteln, die dem aktuellen Mitarbeiter durch den `LeaseAssignmentManager` Vorgang zugewiesen wurden. Die Logik für diesen Vorgang besteht darin, Leasingverträge zu identifizieren, die dem aktuellen Mitarbeiter zugewiesen sind, indem der globale sekundäre Index der Leasingtabelle gelesen wird.

Metrik	Description
ListLeaseKeysForWorker. Zeit	Zeit, den globalen sekundären Index für die Leasing-Tabelle aufzurufen und die Leasingschlüssel abzurufen, die dem aktuellen Worker zugewiesen wurden. Metrikstufe: Detailed Einheiten: Millisekunden
FetchNewLeases. Zeit	Zeit, alle neuen Leasingverträge aus der Leasing-Tabelle abzurufen.

Metrik	Description
	Metrikstufe: Detailed Einheiten: Millisekunden
NewLeasesDiscovered	Gesamtzahl der neuen Leasingverträge, die Arbeitnehmern zugewiesen wurden. Metrikstufe: Detailed Einheiten: Anzahl
Zeit	Zeit, die der `LeaseDiscovery` Vorgang in Anspruch genommen hat. Metrikstufe: Summary Einheiten: Millisekunden
Herzlichen Glückwunsch	Häufigkeit des erfolgreichen Abschlusses der `LeaseDiscovery` - Operation. Metrikstufe: Summary Einheiten: Anzahl
OwnerMismatch	Anzahl der Eigentümer stimmt nicht mit der GSI-Antwort und dem konsistenten Lesen der Leasingtabelle überein. Metrikstufe: Detailed Einheiten: Anzahl

RenewAllLeases

Die Operation `RenewAllLeases` erneuert periodisch alle Shard-Leases, die zu einer bestimmten Worker-Instance gehören.

Metrik	Description
RenewLease. Erfolg	Anzahl der erfolgreichen Lease-Erneuerungen durch den Worker.

Metrik	Description
	Metrikstufe: Detailed Einheiten: Anzahl
RenewLease.Zeit	Zeitbedarf für die Operation der Lease-Erneuerung. Metrikstufe: Detailed Einheiten: Millisekunden
CurrentLeases	Anzahl der Shard-Leases, die einem Worker nach der Erneuerung aller Leases gehören. Metrikstufe: Summary Einheiten: Anzahl
LostLeases	Anzahl der Shard-Leases, die nach einem Versuch zur Erneuerung aller Leases eines Workers verloren gingen. Metrikstufe: Summary Einheiten: Anzahl
Herzlichen Glückwunsch	Häufigkeit, mit der der Vorgang zur Verlängerung des Mietvertrags für den Arbeitnehmer erfolgreich war. Metrikstufe: Summary Einheiten: Anzahl
Zeit	Zeitbedarf für die Erneuerung aller Leases für den Worker. Metrikstufe: Summary Einheiten: Millisekunden

TakeLeases

Die Operation `TakeLeases` sorgt für den Ausgleich aller Datensatzverarbeitungsvorgänge zwischen allen KCL-Workern. Wenn der aktuelle KCL-Worker weniger Shard-Leases als erforderlich hat, werden Shard-Leases von einem anderen, überlasteten, Worker genommen.

Metrik	Description
ListLeases. Erfolg	Häufigkeit, mit der alle Shard-Leases erfolgreich aus der DynamoDB-Tabelle der KCL-Anwendung abgerufen wurden. Metrikstufe: Detailed Einheiten: Anzahl
ListLeases.Zeit	Zeitbedarf für das Abrufen aller Shard-Leases aus der DynamoDB-Tabelle der KCL-Anwendung. Metrikstufe: Detailed Einheiten: Millisekunden
TakeLease.Erfolg	Häufigkeit der erfolgreichen Übernahme von Shard-Leases von anderen KCL-Workern. Metrikstufe: Detailed Einheiten: Anzahl
TakeLease.Zeit	Zeitbedarf für die Aktualisierung der Lease-.Tabelle mit Leases des Workers. Metrikstufe: Detailed Einheiten: Millisekunden
NumWorkers	Gesamtzahl der Worker, wie von einem spezifischen Worker definiert. Metrikstufe: Summary Einheiten: Anzahl

Metrik	Description
NeededLeases	Anzahl der Shard-Leases, die der aktuelle Worker für eine ausgeglichene Shard-Verarbeitungslast benötigt. Metrikstufe: Detailed Einheiten: Anzahl
LeasesToTake	Anzahl der Leases, die der Worker zu übernehmen versuchen wird. Metrikstufe: Detailed Einheiten: Anzahl
TakenLeases	Anzahl der von dem Worker erfolgreich übernommenen Leases. Metrikstufe: Summary Einheiten: Anzahl
TotalLeases	Gesamtzahl der Shards, die die KCL-Anwendung verarbeitet. Metrikstufe: Detailed Einheiten: Anzahl
ExpiredLeases	Gesamtzahl der Shards, die nicht von einem Worker verarbeitet werden, wie von einem spezifischen Worker identifiziert. Metrikstufe: Summary Einheiten: Anzahl
Herzlichen Glückwunsch	Häufigkeit des erfolgreichen Abschlusses der `TakeLeases` -Operation. Metrikstufe: Summary Einheiten: Anzahl

Metrik	Description
Zeit	Zeitbedarf für die `TakeLeases` -Operation für einen Worker. Metrikstufe: Summary Einheiten: Millisekunden

Metriken pro Shard

Diese Metriken werden über einen einzigen Datensatzprozessor aggregiert.

ProcessTask

Die `ProcessTask`-Operation ruft GetRecords mit der aktuellen Iterator-Position auf, um Datensätze aus dem Stream abzurufen, und ruft die Datensatzprozessorfunktion `processRecords` auf.

Metrik	Description
KinesisDataFetcher .getRecords.Success	Anzahl der erfolgreichen `GetRecords` -Operationen pro Kinesis-Datenstrom-Shard. Metrikstufe: Detailed Einheiten: Anzahl
KinesisDataFetcher .getRecords.Time	Zeitbedarf für die `GetRecords` -Operation für den Kinesis-Datenstrom-Shard. Metrikstufe: Detailed Einheiten: Millisekunden
UpdateLease.Erfolg	Anzahl der erfolgreichen Checkpoints durch den Datensatzprozessor für einen bestimmten Shard. Metrikstufe: Detailed Einheiten: Anzahl
UpdateLease.Zeit	Zeitbedarf für jede Checkpoint-Operation eines bestimmten Shards.

Metrik	Description
	Metrikstufe: Detailed Einheiten: Millisekunden
DataBytesProcessed	Gesamtgröße der verarbeiteten Datensätze in Byte bei jedem `ProcessTask` -Aufruf. Metrikstufe: Summary Einheiten: Byte
RecordsProcessed	Anzahl der bei jedem `ProcessTask` -Aufruf verarbeiteten Datensätze. Metrikstufe: Summary Einheiten: Anzahl
ExpiredIterator	Nummer der beim Anruf ExpiredIteratorException `GetRecords` empfangenen Personen. Metrikstufe: Summary Einheiten: Anzahl
MillisBehindLatest	Zeitraum, um den der aktuelle Iterator seit dem letzten Datensatz (Tip) in dem Shard verspätet ist. Dieser Wert ist kleiner oder gleich der Zeitdiffe renz zwischen dem letzten Datensatz in einer Antwort und dem aktuellen Zeitpunkt. Dies ist ein genaueres Bild davon, wie weit ein Shard von der Spitze entfernt ist, als der Vergleich der Zeitstempel im letzten Antwortda tensatz. Dieser Wert bezieht sich auf den letzten Stapel von Datensätz en, nicht auf den Durchschnitt aller Zeitstempel in jedem Datensatz. Metrikstufe: Summary Einheiten: Millisekunden

Metrik	Description
RecordProcessor.processRecords.Time	Zeitbedarf für die `processRecords` -Methode des Datensatzprozessors. Metrikstufe: Summary Einheiten: Millisekunden
Herzlichen Glückwunsch	Anzahl der erfolgreichen Verarbeitungsaufgabenoperationen. Metrikstufe: Summary Einheiten: Anzahl
Zeit	Zeitbedarf für die Verarbeitungsaufgabenoperation. Metrikstufe: Summary Einheiten: Millisekunden

Überwachen Sie die Kinesis Producer Library mit Amazon CloudWatch

Die Amazon Kinesis Producer Library (KPL) für Amazon Kinesis Data Streams veröffentlicht in Ihrem Namen benutzerdefinierte CloudWatch Amazon-Metriken. Sie können sich diese Metriken ansehen, indem Sie zur CloudWatch Konsole navigieren und Benutzerdefinierte Metriken auswählen. Weitere Informationen zu benutzerdefinierten Metriken finden Sie unter Veröffentlichen benutzerdefinierter Metriken im CloudWatch Amazon-Benutzerhandbuch.

Für die CloudWatch von der KPL hochgeladenen Metriken wird eine geringe Gebühr erhoben. Insbesondere fallen Gebühren für Amazon CloudWatch Custom Metrics und Amazon CloudWatch API Requests an. Weitere Informationen finden Sie unter CloudWatch Amazon-Preise. Die Erfassung lokaler Metriken ist nicht mit CloudWatch-Gebühren verbunden.

Themen

- Metriken, Dimensionen und Namespaces
- Metrikebene und Granularität

- Lokaler Zugriff und CloudWatch Amazon-Upload
- Liste der Metriken

Metriken, Dimensionen und Namespaces

Sie können einen Anwendungsnamen festlegen, wenn Sie das KPL starten, das dann beim Hochladen von Metriken als Teil des Namespaces verwendet wird. Dies ist optional. Das KPL bietet einen Standardwert, wenn kein Anwendungsname eingerichtet wurde.

Sie können das KPL auch so konfigurieren, dass beliebige zusätzliche Dimensionen zu den Metriken hinzugefügt werden. Dies ist nützlich, wenn Sie detailliertere Daten in Ihren Metriken haben möchten. CloudWatch Sie können beispielsweise den Host-Namen als Dimension hinzufügen, damit Sie ungleichmäßige Lastverteilungen in der Flotte erkennen. Alle KPL-Konfigurationseinstellungen sind unveränderlich, diese zusätzlichen Dimensionen können nach der Initialisierung der KPL-Instance also nicht mehr geändert werden.

Metrikebene und Granularität

Es gibt zwei Möglichkeiten, um zu steuern, wie viele Metriken zu CloudWatch hochgeladen werden:

Metrikstufe

Dies ist ein grobes Maß für die Wichtigkeit der Metrik. Jede Metrik wird eine Stufe zugewiesen. Wenn Sie eine Ebene festlegen, werden Metriken, deren Stufen darunter liegen, nicht an CloudWatch diese gesendet. Die Stufen sind `NONE`, `SUMMARY` und `DETAILED`. Die Standardeinstellung ist `DETAILED`; das heißt, alle Metriken. `NONE` bedeutet keine Metriken, dieser Stufe werden daher keine Metriken zugewiesen.

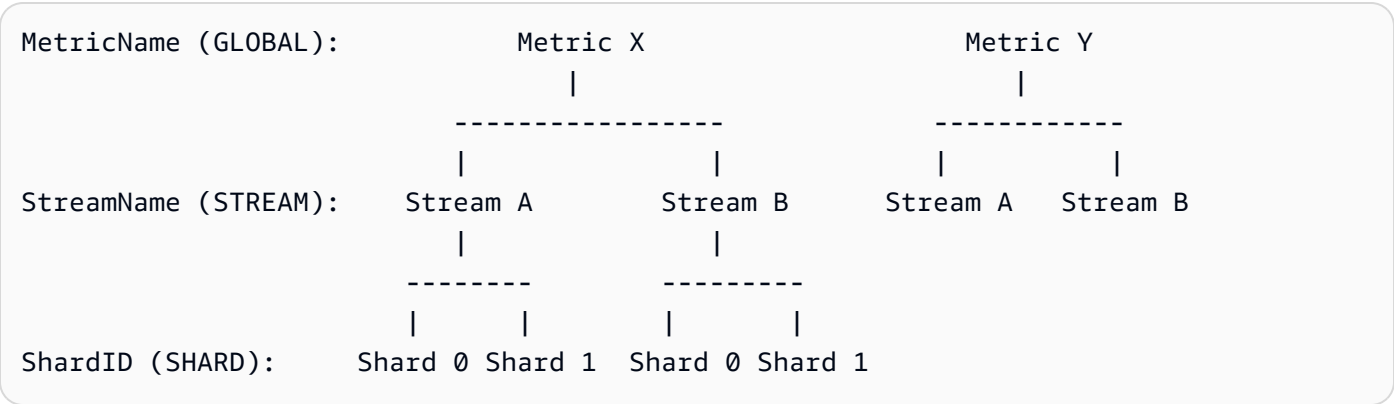
Granularität

Dies steuert, ob deine Metrik mit weiteren Granularitätsstufen ausgegeben werden soll. Die Stufen sind `GLOBAL`, `STREAM` und `SHARD`. Die Standardeinstellung ist `SHARD`mit den detailliertesten Metriken.

Bei Auswahl von `SHARD` werden Metriken mit dem Stream-Namen und der Shard-ID als Dimensionen ausgegeben. Dazu wird die gleiche Metrik auch nur mit der Stream-Name-Dimension und ohne die Stream-Name-Dimension ausgegeben. Das bedeutet, dass für eine bestimmte Metrik zwei Streams mit jeweils zwei Shards sieben CloudWatch Metriken erzeugen: eine für jeden Shard, eine für jeden Stream und eine insgesamt. Alle beschreiben dieselben

Statistiken, aber auf unterschiedlichen Granularitätsebenen. Eine Illustration finden Sie im folgenden Diagramm.

Die unterschiedlichen Granularitätsebenen bilden eine Hierarchie, und alle Metriken in dem System bilden einen Baum, dessen Wurzel die Metriknamen bilden:



Nicht alle Metriken sind auf Shard-Ebene verfügbar; einige sind ihrer Natur nach global oder auf die Stream-Ebene beschränkt. Sie werden nicht auf Shard-Ebene produziert, auch wenn Sie Kennzahlen auf Shard-Ebene aktiviert haben (`Metric Y` im Diagramm oben).

Wenn Sie eine zusätzliche Dimension angeben, müssen Sie Werte für angeben. `tuple:<DimensionName, DimensionValue, Granularity>` Die Granularität legt fest, wo die benutzerdefinierte Dimension in die Hierarchie eingefügt wird: `GLOBAL` bedeutet, dass die zusätzliche Dimension hinter dem Kennzahlnamen eingefügt wird, `STREAM` bedeutet, dass sie hinter dem Stream-Namen und `SHARD`, dass sie hinter der Shard-ID eingefügt wird. Wenn mehrere Dimensionen pro Granularitätsstufe angegeben sind, werden sie in der angegebenen Reihenfolge eingefügt.

Lokaler Zugriff und CloudWatch Amazon-Upload

Metriken für die aktuelle KPL-Instance sind lokal in Echtzeit verfügbar; Sie können dazu jederzeit das KPL abfragen. Die KPL berechnet lokal die Summe, den Durchschnitt, das Minimum, das Maximum und die Anzahl jeder Metrik, wie in. CloudWatch

Sie erhalten statistische Werte, die seit dem Start des Programms bis zum aktuellen Zeitpunkt kumuliert wurden, oder Sie können ein laufendes Zeitfenster über die letzten N Sekunden verwenden, wobei N eine ganze Zahl zwischen 1 und 60 ist.

Alle Metriken können hochgeladen werden. CloudWatch Dies ist besonders nützlich für die Aggregierung von Daten über mehrere Hosts sowie Überwachungs- und Alarmvorgänge. Diese Funktionalität ist nicht lokal verfügbar.

Wie zuvor beschrieben, können Sie wählen, welche Metriken mit der Metrik-Ebene und Granularität hochgeladen werden sollen. Metriken, die nicht hochgeladen wurde, sind lokal verfügbar.

Das Hochladen einzelner Datenpunkte ist nicht sinnvoll, da dies zu Millionen von Uploads pro Sekunde führen könnte, wenn der Datenverkehr hoch ist. Aus diesem Grund aggregiert die KPL Metriken lokal in 1-Minuten-Buckets und lädt ein Statistikobjekt CloudWatch einmal pro Minute pro aktivierter Metrik hoch.

Liste der Metriken

Metrik	Description
`UserRecordsReceived`	Anzahl der logischen Benutzerdatensätze, die vom KPL-Core für Put-Operationen erhalten wurden. Auf Shard-Ebene nicht verfügbar. Metrikstufe: Detailed Einheit: Anzahl
`UserRecordsPending`	Regelmäßige Stichprobe, wie viele Benutzer Datensätze derzeit ausstehen. Ein Datensatz gilt als ausstehend, wenn er entweder gerade gepuffert ist und darauf wartet, gesendet zu werden, oder wenn er gerade an den Backend-Service gesendet wird. Auf Shard-Ebene nicht verfügbar. Das KPL bietet eine dedizierte Methode zum Abrufen dieser Metrik auf globaler Ebene, damit Kunden ihre Put-Rate verwalten können. Metrikstufe: Detailed Einheit: Anzahl
`UserRecordsPut`	Anzahl der logischen Benutzerdatensätze mit erfolgreichen Put-Operationen.

Metrik	Description
	Das KPL gibt eine Null für fehlgeschlagene Datensätze aus. Dadurch kann der Durchschnittswert die Erfolgsrate, die Anzahl die Gesamtzahl der Versuche und die Differenz zwischen beiden die Anzahl der Fehlschläge angeben. Metrikstufe: Summary Einheit: Anzahl
`UserRecordsDataPut`	Bytes in den logischen Benutzer-Datensätzen mit erfolgreichen Put-Operationen. Metrikstufe: Detailed Einheit: Byte
`KinesisRecordsPut`	Anzahl der Datensätze der Kinesis Data Streams mit erfolgreichen Put-Operationen (jeder Datensatz der Kinesis Data Streams kann mehrere Benutzerdatensätze enthalten). Das KPL gibt eine Null für fehlgeschlagene Datensätze aus. Dadurch kann der Durchschnittswert die Erfolgsrate, die Anzahl die Gesamtzahl der Versuche und die Differenz zwischen beiden die Anzahl der Fehlschläge angeben. Metrikstufe: Summary Einheit: Anzahl
`KinesisRecordsDataPut`	Bytes in den Datensätzen der Kinesis Data Streams. Metrikstufe: Detailed Einheit: Byte

Metrik	Description
`ErrorsByCode`	Anzahl der einzelnen Arten von Fehlercodes. Dies führt eine weitere Dimension von `ErrorCode` zusätzlich zu den normalen Dimension en ein, wie z. B. `StreamName` und `ShardId`. Nicht jeder Fehler kann zu einem Shard verfolgt werden. Die Fehler, die nicht verfolgt werden können, werden nur auf Stream- oder globaler Ebene ausgegebe n. Diese Metrik erfasst Informationen zu Dingen wie Drosselung, Änderungen der Shard-Zuordnung, internen Fehlern, nicht verfügbaren Services, Zeitüberschreitungen u. dgl. API-Fehler der Kinesis Data Streams werden einmal pro Datensatz der Kinesis Data Streams gezählt. Mehrere Benutzerdatensätze innerhalb eines Datensatzes von Kinesis Data Streams erzeugen keine Mehrfachz ählungen. Metrikstufe: Summary Einheit: Anzahl
`AllErrors`	Dies wird von den gleichen Fehlern ausgelöst wie Fehler nach Code, jedoch ohne Unterscheidung zwischen den Typen. Dies ist nützlich zur allgemeinen Überwachung der Fehlerrate, ohne dass eine manuelle Summe der Anzahlen der verschiedenen Arten von Fehlern erforderlich ist. Metrikstufe: Summary Einheit: Anzahl

Metrik	Description
`RetriesPerRecord`	Anzahl der pro Benutzerdatensatz durchgeführten erneuten Versuche. Für Datensätze, die bei einem Versuch erfolgreich waren, wird Null ausgegeben. Die Daten werden in dem Moment ausgegeben, in dem ein Benutzerdatensatz abgeschlossen wird (wenn er entweder erfolgreich war, oder wenn kein Wiederholungsversuch mehr möglich ist). Wenn der Datensatz einen großen Wert hat, kann time-to-live es bei dieser Metrik zu einer erheblichen Verzögerung kommen. Metrikstufe: Detailed Einheit: Anzahl
`BufferingTime`	Der Zeitraum zwischen dem Eingang eines Benutzerdatensatzes beim KPL und seinem Abgang zum Backend. Diese Informationen werden an den Benutzer pro Datensatz zurückgegeben, sind jedoch auch als aggregierte Statistik verfügbar. Metrikstufe: Summary Einheit: Millisekunden
`Request Time`	Die Zeit zum Ausführen von `PutRecordsRequests` . Metrikstufe: Detailed Einheit: Millisekunden
`User Records per Kinesis Record`	Die Anzahl der logischen Benutzerdatensätze, die in einem einzigen Datensatz der Kinesis Data Streams zusammengefasst sind. Metrikstufe: Detailed Einheit: Anzahl

Metrik	Description
`Amazon Kinesis Records per PutRecordsRequest`	Die Anzahl der Datensätze von Kinesis Data Streams, die zu einem einzigen `PutRecordsRequest` aggregiert wurden. Auf Shard-Ebene nicht verfügbar. Metrikstufe: Detailed Einheit: Anzahl
`User Records per PutRecordsRequest`	Die Gesamtanzahl der Benutzerdatensätze in einer `PutRecordsRequest` . Diese Zahl entspricht ungefähr dem Produkt der beiden vorherigen Metriken. Auf Shard-Ebene nicht verfügbar. Metrikstufe: Detailed Einheit: Anzahl

Sicherheit in Amazon Kinesis Data Streams

Cloud-Sicherheit hat AWS höchste Priorität. Als AWS Kunde profitieren Sie von Rechenzentren und Netzwerkarchitekturen, die darauf ausgelegt sind, die Anforderungen der sicherheitssensibelsten Unternehmen zu erfüllen.

Sicherheit ist eine gemeinsame AWS Verantwortung von Ihnen und Ihnen. Das Modell der geteilten Verantwortung beschreibt dies als Sicherheit der Cloud und Sicherheit in der Cloud:

- Sicherheit der Cloud — AWS ist verantwortlich für den Schutz der Infrastruktur, die AWS Dienste in der AWS Cloud ausführt. AWS bietet Ihnen auch Dienste, die Sie sicher nutzen können. Auditoren von Drittanbietern testen und überprüfen die Effektivität unserer Sicherheitsmaßnahmen im Rahmen der AWS -Compliance-Programme regelmäßig. Weitere Informationen zu den Compliance-Programmen für Kinesis Data Streams finden Sie unter Durch das Compliance-Programm abgedeckte AWS -Services.
- Sicherheit in der Cloud — Ihre Verantwortung richtet sich nach dem AWS Dienst, den Sie nutzen. In Ihre Verantwortung fallen außerdem weitere Faktoren, wie z. B. die Vertraulichkeit der Daten, die Anforderungen Ihrer Organisation sowie geltende Gesetze und Vorschriften.

Diese Dokumentation hilft Ihnen zu verstehen, wie Sie das Modell der geteilten Verantwortung bei der Verwendung von Kinesis Data Streams einsetzen können. Die folgenden Themen veranschaulichen, wie Sie Kinesis Data Streams zur Erfüllung Ihrer Sicherheits- und Compliance-Ziele konfigurieren können. Sie lernen auch, wie Sie andere AWS Dienste nutzen können, die Ihnen helfen, Ihre Kinesis Data Streams Streams-Ressourcen zu überwachen und zu sichern.

Topics

- Datenschutz in Amazon Kinesis Data Streams
- Steuern des Zugriffs auf Amazon Kinesis Data Streams Streams-Ressourcen mithilfe von IAM
- Konformitätsprüfung für Amazon Kinesis Data Streams
- Resilienz in Amazon Kinesis Data Streams
- Infrastruktursicherheit in Amazon Kinesis Data Streams
- Bewährte Sicherheitsmethoden für Kinesis Data Streams

Datenschutz in Amazon Kinesis Data Streams

Die serverseitige Verschlüsselung mit AWS Key Management Service (AWS KMS) -Schlüsseln macht es Ihnen leicht, strenge Datenverwaltungsanforderungen zu erfüllen, indem Sie Ihre ruhenden Daten in Amazon Kinesis Data Streams verschlüsseln.

> **Note**
>
> Wenn Sie für den Zugriff AWS über eine Befehlszeilenschnittstelle oder eine API FIPS 140-2-validierte kryptografische Module benötigen, verwenden Sie einen FIPS-Endpunkt. Weitere Informationen über verfügbare FIPS-Endpunkte finden Sie unter Federal Information Processing Standard (FIPS) 140-2.

Themen

- Was ist serverseitige Verschlüsselung für Kinesis Data Streams?
- Überlegungen zu Kosten, Regionen und Leistung
- Wie fange ich mit serverseitiger Verschlüsselung an?
- Erstellen und verwenden Sie benutzergenerierte KMS-Schlüssel
- Berechtigungen zur Verwendung von benutzergenerierten KMS-Schlüsseln
- Überprüfen Sie die KMS-Schlüsselberechtigungen und beheben Sie Fehler
- Verwenden Sie Amazon Kinesis Data Streams mit VPC-Endpunkten mit Schnittstellen

Was ist serverseitige Verschlüsselung für Kinesis Data Streams?

Serverseitige Verschlüsselung ist eine Funktion in Amazon Kinesis Data Streams, die Daten automatisch verschlüsselt, bevor sie gespeichert werden, indem ein von Ihnen festgelegter AWS KMS Customer Master Key (CMK) verwendet wird. Die Daten werden verschlüsselt, bevor sie in die Speicherschicht des Kinesis-Streams geschrieben werden. Nach Abruf aus dem Speicher werden Sie entschlüsselt. Dadurch sind Ihre ruhenden Daten innerhalb des Services Kinesis Data Streams verschlüsselt. Sie erfüllen so die strengen regulatorischen Anforderungen und erhöhen die Sicherheit Ihrer Daten.

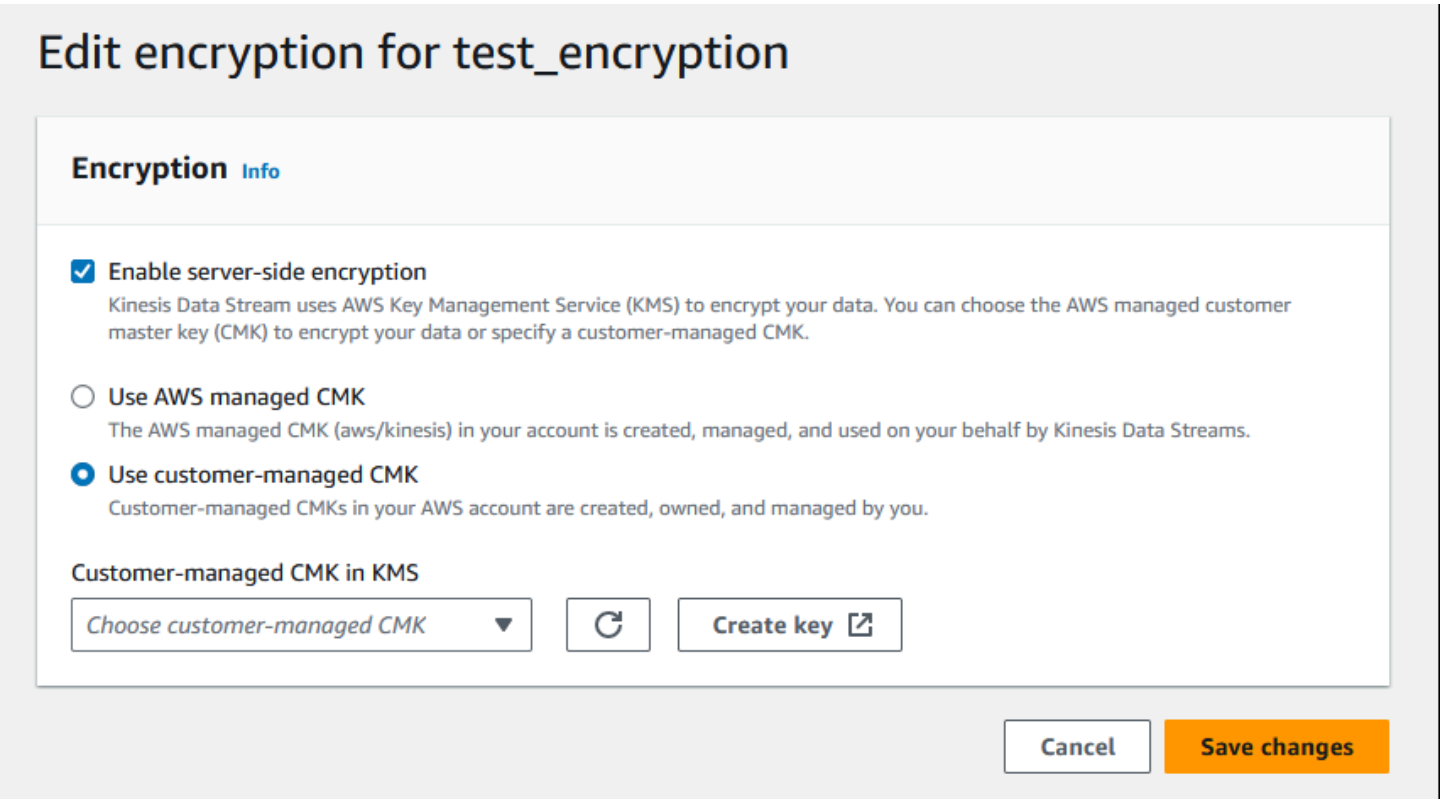
Durch die serverseitige Verschlüsselung müssen Ihre Kinesis-Stream-Produzenten und -Verbraucher keine Masterschlüssel oder kryptographische Operationen verwalten. Ihre Daten werden automatisch

verschlüsselt, wenn sie den Kinesis Data Streams Streams-Dienst betreten und verlassen, sodass Ihre Daten im Ruhezustand verschlüsselt sind. AWS KMS stellt alle Masterschlüssel bereit, die von der serverseitigen Verschlüsselungsfunktion verwendet werden. AWS KMS macht es einfach, ein CMK für Kinesis zu verwenden, das von AWS einem benutzerdefinierten AWS KMS CMK oder einem in den Service importierten Masterschlüssel verwaltet wird. AWS KMS

> **Note**
>
> Bei der serverseitigen Verschlüsselung werden eingehende Daten nur dann verschlüsselt, wenn die Funktion aktiviert ist. Bereits in einem nicht verschlüsselten Stream vorhandene Daten werden nach Aktivierung der serverseitigen Verschlüsselung nicht verschlüsselt.

Wenn Sie Ihre Datenströme verschlüsseln und den Zugriff mit anderen Principals teilen, müssen Sie die entsprechenden Berechtigungen sowohl in der Schlüsselrichtlinie für den Schlüssel als auch in den IAM-Richtlinien für das AWS KMS externe Konto erteilen. Weitere Informationen finden Sie unter Benutzern in anderen Konten die Verwendung eines KMS-Schlüssels erlauben.

Wenn Sie die serverseitige Verschlüsselung für einen Datenstream mit AWS verwaltetem KMS-Schlüssel aktiviert haben und den Zugriff über eine Ressourcenrichtlinie gemeinsam nutzen möchten, müssen Sie zur Verwendung des vom Kunden verwalteten Schlüssels (CMK) wechseln, wie im Folgenden dargestellt:



Darüber hinaus müssen Sie Ihren freigebenden Prinzipal-Entitäten Zugriff auf Ihren CMK gewähren, indem Sie die kontoübergreifenden KMS-Freigabe-Funktionen nutzen. Stellen Sie sicher, dass Sie auch die IAM-Richtlinien für die freigebenden Prinzipal-Entitäten ändern. Weitere Informationen finden Sie unter Benutzern in anderen Konten die Verwendung eines KMS-Schlüssels erlauben.

Überlegungen zu Kosten, Regionen und Leistung

Wenn Sie serverseitige Verschlüsselung anwenden, fallen Kosten für AWS KMS API-Nutzung und Schlüssel an. Im Gegensatz zu benutzerdefinierten KMS-Masterschlüsseln wird der benutzerdefinierte `(Default) aws/kinesis`-Masterschlüssel (Customer Master Key, CMK) kostenlos zur Verfügung gestellt. Sie müssen jedoch weiterhin die Kosten für die API-Nutzung tragen, die Amazon Kinesis Data Streams für Sie verursacht.

Kosten für die API-Nutzung fallen für alle CMKs an, einschließlich benutzerdefinierten. Kinesis Data Streams ruft AWS KMS ungefähr alle fünf Minuten ab, wenn der Datenschlüssel gewechselt wird. In einem Monat mit 30 Tagen sollten die Gesamtkosten für AWS KMS API-Aufrufe, die von einem Kinesis-Stream initiiert werden, weniger als ein paar Dollar betragen. Diese Kosten steigen mit der Anzahl der Benutzeranmeldedaten, die Sie bei Ihren Datenproduzenten und -verbrauchern verwenden, da für jede Benutzeranmeldeberechtigung ein eindeutiger API-Aufruf erforderlich ist. AWS KMS Wenn Sie eine IAM-Rolle für die Authentifizierung verwenden, führt jeder

Rollenübernahmeaufruf zu eindeutigen Benutzeranmeldeinformationen. Um KMS-Kosten zu sparen, können Sie Benutzeranmeldeinformationen zwischenspeichern, die vom Rollenübernahmeaufruf zurückgegeben werden.

Im Folgenden werden die Kosten pro Ressource aufgeführt:

Schlüssel

- Das CMK für Kinesis, das von AWS (alias =`aws/kinesis`) verwaltet wird, ist kostenlos.
- Von Benutzern generierte KMS-Schlüssel unterliegen den KMS-Schlüssel-Kosten. Weitere Informationen finden Sie unter AWS Key Management Service – Preise.

Kosten für die API-Nutzung fallen für alle CMKs an, einschließlich benutzerdefinierten. Kinesis Data Streams ruft KMS ungefähr alle fünf Minuten ab, wenn der Datenschlüssel gewechselt wird. In einem Monat mit 30 Tagen sollten die Gesamtkosten für KMS-API-Aufrufe, die von einem Kinesis-Datenstrom initiiert werden, nur wenige Dollar betragen. Bitte beachten Sie, dass diese Kosten mit der Anzahl der Benutzeranmeldedaten, die Sie bei Ihren Datenproduzenten und -verbrauchern verwenden, steigen, da für jede Benutzeranmeldeberechtigung ein eindeutiger API-Aufruf an KMS erforderlich ist. AWS Wenn Sie die IAM-Rolle für die Authentifizierung verwenden, assume-role-call führt jede Rolle zu eindeutigen Benutzeranmeldeinformationen, und Sie sollten die von der zurückgegebenen Benutzeranmeldeinformationen zwischenspeichern, um assume-role-call KMS-Kosten zu sparen.

Verwendung der KMS-API

Für jeden verschlüsselten Stream ruft der Kinesis-Dienst den AWS KMS Dienst etwa 12 Mal alle 5 Minuten auf, wenn aus TIP gelesen und ein einziger account/user IAM-Zugriffsschlüssel für alle Leser und Autoren verwendet wird. Wenn Sie nicht aus TIP lesen, kann dies zu höheren Serviceaufrufen führen. AWS KMS API-Anfragen zur Generierung neuer Datenverschlüsselungsschlüssel sind mit AWS KMS Nutzungskosten verbunden. Weitere Informationen finden Sie unter Preise für AWS Key Management Service – Verwendung.

Verfügbarkeit serverseitiger Verschlüsselung nach Regionen

Derzeit ist die serverseitige Verschlüsselung von Kinesis-Streams in allen Regionen verfügbar, die für Kinesis Data Streams unterstützt werden, einschließlich AWS GovCloud (USA West) und den Regionen China. Weitere Informationen zu unterstützten Regionen für Kinesis Data Streams finden Sie unter https://docs.aws.amazon.com/general/latest/gr/ak.html.

Performanceaspekte

Aufgrund des Serviceaufwands bei der Verschlüsselung erhöht die serverseitige Verschlüsselung die typische Latenzzeit von `PutRecord`, `PutRecords` und `GetRecords` um 100µs.

Wie fange ich mit serverseitiger Verschlüsselung an?

Der einfachste Weg, mit serverseitiger Verschlüsselung zu beginnen, ist die Verwendung des AWS-Managementkonsole und des Amazon Kinesis KMS Service Keys,. `aws/kinesis`

Im Folgenden wird die Aktivierung der serverseitigen Verschlüsselung für einen Kinesis-Stream beschrieben.

So aktivieren Sie die serverseitige Verschlüsselung für einen Kinesis-Stream

1. Melden Sie sich bei der Amazon Kinesis Data Streams Streams-Konsole an AWS-Managementkonsole und öffnen Sie sie.
2. Wählen oder erstellen Sie einen Kinesis-Stream in der AWS-Managementkonsole.
3. Wählen Sie die Registerkarte Details aus.
4. Wählen Sie unter Server-side encryption (Serverseitige Verschlüsselung) die Option edit (Bearbeiten) aus.
5. Wenn Sie einen benutzergenerierten KMS-Masterschlüssel verwenden möchten, stellen Sie sicher, dass der (standardmäßige) aws/kinesis-KMS-Masterschlüssel ausgewählt ist. Dies ist die KMS-Masterschlüssel, der vom Kinesis-Service generiert wurde. Wählen Sie Enabled (Aktiviert) und anschließend Save (Speichern) aus.

   > **Note**
   >
   > Der Standard-Hauptschlüssel für den Kinesis-Service ist kostenlos. Für die API-Aufrufe von Kinesis an den AWS KMS Service fallen jedoch KMS-Nutzungskosten an.

6. Der Stream ist vorübergehend im Zustand ausstehend. Sobald sich der Stream in einem aktiven Zustand befindet und die Verschlüsselung aktiviert ist, werden alle in den Stream eingeleiteten Daten mit dem von Ihnen ausgewählten KMS-Masterschlüssel verschlüsselt.
7. Um die serverseitige Verschlüsselung zu deaktivieren, wählen Sie in der unter Serverseitige Verschlüsselung die Option Deaktiviert und wählen Sie dann AWS-Managementkonsole Speichern aus.

Erstellen und verwenden Sie benutzergenerierte KMS-Schlüssel

In diesem Abschnitt wird beschrieben, wie Sie Ihre eigenen KMS-Schlüssel erstellen und verwenden, anstatt den von Amazon Kinesis verwalteten Masterschlüssel zu verwenden.

Erstellen Sie benutzergenerierte KMS-Schlüssel

Anweisungen zum Erstellen eigener Schlüssel finden Sie unter Creating Keys im AWS Key Management Service Developer Guide. Nachdem Sie Schlüssel für Ihr Konto erstellt haben, gibt der Service Kinesis Data Streams diese Schlüssel über die Liste KMS-Masterschlüsse zurück.

Verwenden Sie benutzergenerierte KMS-Schlüssel

Nachdem Ihren Verbrauchern, Produzenten und Administratoren die richtigen Berechtigungen zugewiesen wurden, können Sie benutzerdefinierte KMS-Schlüssel in Ihrem eigenen AWS Konto oder einem anderen AWS Konto verwenden. Alle KMS-Masterschlüssel in Ihrem Konto werden in der in der Liste für KMS-Masterschlüssel AWS-Managementkonsole angezeigt.

Für die Nutzung benutzerdefinierter KMS-Schlüssel eines anderen Kontos benötigen Sie entsprechende Berechtigungen. Darüber hinaus müssen Sie den ARN des KMS-Masterschlüssels in der AWS-Managementkonsole im ARN-Eingabefeld eingeben.

Berechtigungen zur Verwendung von benutzergenerierten KMS-Schlüsseln

Bevor Sie die serverseitige Verschlüsselung mit einem benutzergenerierten KMS-Schlüssel verwenden können, müssen Sie AWS KMS Schlüsselrichtlinien konfigurieren, um die Verschlüsselung von Streams sowie die Verschlüsselung und Entschlüsselung von Stream-Datensätzen zu ermöglichen. Beispiele und weitere Informationen zu AWS KMS Berechtigungen finden Sie unter AWS KMS-API-Berechtigungen: Referenz zu Aktionen und Ressourcen.

> **Note**
>
> Für die Nutzung des Standard-Serviceschlüssels für die Verschlüsselung sind keine benutzerdefinierten IAM-Berechtigungen erforderlich.

Ehe Sie benutzergenerierte KMS-Masterschlüssel verwenden, müssen Sie sicherstellen, dass Ihre Kinesis-Stream-Produzenten und -Konsumenten (IAM-Prinzipale) Benutzer im Sinne der Richtlinie

für KMS-Masterschlüssel sind. Andernfalls schlägt das Schreiben in und das Lesen aus dem Stream fehl. Dies kann zu einem Datenverlust, einer verspäteten Verarbeitung oder abgestürzten Anwendungen führen. Sie können Berechtigungen für KMS-Schlüssel mit IAM-Richtlinien verwalten. Weitere Informationen finden Sie unter Verwenden von IAM-Richtlinien mit AWS KMS.

Kinesis Data Streams Streams-Verschlüsselungskontext

Wenn Amazon Kinesis Data Streams in AWS KMS Ihrem Namen anruft, leitet es einen Verschlüsselungskontext an AWS KMS , der als Bedingung für die Autorisierung in wichtigen Richtlinien und Zuschüssen verwendet werden kann. Kinesis Data Streams verwendet den Stream-ARN als Verschlüsselungskontext in allen AWS KMS Aufrufen.

```
"encryptionContext": {
    "aws:kinesis:arn": "arn:aws:kinesis:region:account-id:stream/stream-name"
}
```

Sie können den Verschlüsselungskontext verwenden, um die Verwendung Ihres KMS-Schlüssels in Prüfaufzeichnungen und Protokollen zu identifizieren. Er erscheint auch im Klartext in Protokollen, wie AWS CloudTrail z.

Um die Verwendung Ihres KMS-Schlüssels auf Anfragen von Kinesis Data Streams für einen bestimmten Stream zu beschränken, verwenden Sie den `kms:EncryptionContext:aws:kinesis:arn` Bedingungsschlüssel in der KMS-Schlüsselrichtlinie oder IAM-Richtlinie.

Beispiel für Herstellerberechtigungen

Die Kinesis-Stream-Produzenten benötigen die Berechtigung `kms:GenerateDataKey`.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "kms:GenerateDataKey"
        ],
```

```
        "Resource": "arn:aws:kms:us-
west-2:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    {
        "Effect": "Allow",
        "Action": [
            "kinesis:PutRecord",
            "kinesis:PutRecords"
        ],
        "Resource": "arn:aws:kinesis:*:123456789012:MyStream"
    }
  ]
}
```

Beispiel für Verbraucherberechtigungen

Ihre Kinesis-Stream-Konsumenten benötigen die Berechtigung `kms:Decrypt`.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "kms:Decrypt"
        ],
        "Resource": "arn:aws:kms:us-
west-2:123456789012:key/1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    {
        "Effect": "Allow",
        "Action": [
            "kinesis:GetRecords",
            "kinesis:DescribeStream"
        ],
        "Resource": "arn:aws:kinesis:*:123456789012:MyStream"
    }
  ]
}
```

Amazon Managed Service für Apache Flink und AWS Lambda verwenden Sie Rollen, um Kinesis-Streams zu nutzen. Stellen Sie sicher, dass Sie die `kms:Decrypt`-Berechtigung für die Rollen erteilen, die diese Konsumenten innehaben.

Stream-Administratorberechtigungen

Kinesis-Stream-Administratoren benötigen eine Autorisierung für den Aufruf von `kms:List*` und `kms:DescribeKey*`.

Überprüfen Sie die KMS-Schlüsselberechtigungen und beheben Sie Fehler

Nachdem Sie die Verschlüsselung für einen Kinesis-Stream aktiviert haben, empfehlen wir Ihnen, den Erfolg Ihrer `putRecordputRecords`, und `getRecords` -Anrufe anhand der folgenden CloudWatch Amazon-Metriken zu überwachen:

- `PutRecord.Success`
- `PutRecords.Success`
- `GetRecords.Success`

Weitere Informationen finden Sie unter Kinesis Data Streams überwachen.

Verwenden Sie Amazon Kinesis Data Streams mit VPC-Endpunkten mit Schnittstellen

Sie können einen VPC-Schnittstellen-Endpunkt verwenden, um zu verhindern, dass der Datenverkehr zwischen Ihrer Amazon VPC und Kinesis Data Streams das Amazon-Netzwerk verlässt. Schnittstellen-VPC-Endpunkte benötigen kein Internet-Gateway, kein NAT-Gerät, keine VPN-Verbindung oder Direct Connect Verbindung. Interface VPC-Endpoints basieren auf einer AWS Technologie AWS PrivateLink, die private Kommunikation zwischen AWS Services über eine elastic network interface mit Private IPs in Ihrer Amazon VPC ermöglicht. Weitere Informationen finden Sie unter Amazon Virtual Private Cloud und Interface VPC Endpoints ()AWS PrivateLink.

Themen

- Verwenden Sie Schnittstellen-VPC-Endpunkte für Kinesis Data Streams
- Steuern Sie den Zugriff auf VPC-Endpunkte für Kinesis Data Streams
- Verfügbarkeit von VPC-Endpunktrichtlinien für Kinesis Data Streams

Verwenden Sie Schnittstellen-VPC-Endpunkte für Kinesis Data Streams

Zu Beginn müssen Sie die Einstellungen für Ihre Streams, Produzenten oder Verbraucher nicht ändern. Erstellen Sie einen VPC-Schnittstellen-Endpunkt für Ihre Kinesis Data Streams, um den Datenverkehr von und zu Ihren Amazon VPC-Ressourcen über den Schnittstellen-VPC-Endpunkt zu starten. VPC-Endpunkte mit FIPS-fähiger Schnittstelle sind für US-Regionen verfügbar. Weitere Informationen finden Sie unter Erstellen eines Schnittstellenendpunkts.

Die Amazon Kinesis Producer Library (KPL) und die Kinesis Consumer Library (KCL) rufen AWS Dienste wie Amazon CloudWatch und Amazon DynamoDB entweder über öffentliche Endpunkte oder VPC-Endpunkte mit privaten Schnittstellen auf, je nachdem, welche verwendet werden. Wenn Ihre KCL-Anwendung beispielsweise in einer VPC mit DynamoDB-Schnittstelle und aktivierten VPC-Endpunkten ausgeführt wird, fließen Aufrufe zwischen DynamoDB und Ihrer KCL-Anwendung über den VPC-Endpunkt der Schnittstelle.

Steuern Sie den Zugriff auf VPC-Endpunkte für Kinesis Data Streams

Mit VPC-Endpunktrichtlinien können Sie den Zugriff steuern, indem Sie entweder eine Richtlinie an einen VPC-Endpunkt anhängen oder indem Sie zusätzliche Felder in einer Richtlinie verwenden, die einem IAM-Benutzer, einer Gruppe oder einer Rolle zugeordnet ist, um den Zugriff auf den angegebenen VPC-Endpunkt zu beschränken. Verwenden Sie diese Richtlinien, um den Zugriff auf bestimmte Streams auf einen bestimmten VPC-Endpunkt zu beschränken, wenn Sie sie zusammen mit den IAM-Richtlinien verwenden, um nur Zugriff auf Kinesis-Datenstream-Aktionen über den angegebenen VPC-Endpunkt zu gewähren.

Es folgen Beispiele für Endpunktrichtlinien für den Zugriff auf Kinesis-Daten-Streams.

- Beispiel für eine VPC-Richtlinie: schreibgeschützter Zugriff – diese Beispielrichtlinie kann einem VPC-Endpunkt zugeordnet werden. Weitere Informationen finden Sie unter Steuern des Zugriffs auf Amazon VPC-Ressourcen. Sie beschränkt Aktionen auf das Auflisten und Beschreiben eines Kinesis-Daten-Streams über den VPC-Endpunkt, dem er zugeordnet ist.

```
{
  "Statement": [
    {
      "Sid": "ReadOnly",
      "Principal": "*",
      "Action": [
        "kinesis:List*",
```

```
        "kinesis:Describe*"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
  ]
}
```

- Beispiel für eine VPC-Richtlinie: Beschränken des Zugriffs auf einen bestimmten Kinesis-Daten-Stream – diese Beispielrichtlinie kann einem VPC-Endpunkt zugeordnet werden. Sie schränkt den Zugriff auf einen bestimmten Daten-Stream über den VPC-Endpunkt ein, dem er zugeordnet ist.

```
{
  "Statement": [
    {
      "Sid": "AccessToSpecificDataStream",
      "Principal": "*",
      "Action": "kinesis:*",
      "Effect": "Allow",
      "Resource": "arn:aws:kinesis:us-east-1:123456789012:stream/MyStream"
    }
  ]
}
```

- Beispiel für eine IAM-Richtlinie: Beschränken Sie den Zugriff auf einen bestimmten Stream nur von einem bestimmten VPC-Endpunkt aus. Diese Beispielrichtlinie kann einem IAM-Benutzer, einer Rolle oder einer Gruppe zugewiesen werden. Sie beschränkt den Zugriff auf einen bestimmten Kinesis-Daten-Stream auf einen bestimmten VPC-Endpunkt.

  JSON

```
{
   "Version":"2012-10-17",
   "Statement": [
      {
         "Sid": "AccessFromSpecificEndpoint",
         "Action": "kinesis:*",
         "Effect": "Deny",
         "Resource": "arn:aws:kinesis:us-east-1:123456789012:stream/MyStream",
```

```
        "Condition": { "StringNotEquals" : { "aws:sourceVpce":
 "vpce-11aa22bb" } }
      }
   ]
}
```

Verfügbarkeit von VPC-Endpunktrichtlinien für Kinesis Data Streams

VPC-Endpunkte mit Kinesis Data Streams Streams-Schnittstelle und Richtlinien werden in den folgenden Regionen unterstützt:

- Europa (Paris)
- Europa (Irland)
- USA Ost (Nord-Virginia)
- Europa (Stockholm)
- USA Ost (Ohio)
- Europa (Frankfurt)
- Südamerika (São Paulo)
- Europa (London)
- Asien-Pazifik (Tokio)
- USA West (Nordkalifornien)
- Asien-Pazifik (Singapur)
- Asien-Pazifik (Sydney)
- China (Peking)
- China (Ningxia)
- Asia Pacific (Hong Kong)
- Middle East (Bahrain)
- Naher Osten (VAE)
- Europa (Milan)
- Afrika (Kapstadt)
- Asien-Pazifik (Mumbai)
- Asien-Pazifik (Seoul)

- Kanada (Zentral)
- USA West (Oregon) außer usw2-az4
- AWS GovCloud (USA, Ost)
- AWS GovCloud (US-West)
- Asien-Pazifik (Osaka)
- Europa (Zürich)
- Asien-Pazifik (Hyderabad)

Steuern des Zugriffs auf Amazon Kinesis Data Streams Streams-Ressourcen mithilfe von IAM

AWS Identity and Access Management (IAM) ermöglicht Ihnen Folgendes:

- Erstellen Sie Benutzer und Gruppen unter Ihrem Konto AWS
- Weisen Sie jedem Benutzer unter Ihrem AWS Konto eindeutige Sicherheitsanmeldedaten zu
- Kontrollieren Sie die Berechtigungen der einzelnen Benutzer zur Ausführung von Aufgaben mithilfe von AWS Ressourcen
- Erlauben Sie den Benutzern eines anderen AWS Kontos, Ihre AWS Ressourcen gemeinsam zu nutzen
- Erstellen Sie Rollen für Ihr AWS Konto und definieren Sie die Benutzer oder Dienste, die diese Rollen übernehmen können
- Verwenden Sie bestehende Identitäten für Ihr Unternehmen, um Berechtigungen zur Ausführung von Aufgaben unter Verwendung von AWS Ressourcen zu erteilen

Wenn Sie IAM zusammen mit Kinesis Data Streams verwenden, können Sie steuern, ob Benutzer im Unternehmen Aufgaben mit bestimmten API-Aktionen von Kinesis Data Streams ausführen und spezifische AWS -Ressourcen verwenden können.

Wenn Sie eine Anwendung mithilfe der Kinesis Client Library (KCL) entwickeln, muss Ihre Richtlinie Berechtigungen für Amazon DynamoDB und Amazon beinhalten. Die KCL verwendet DynamoDB CloudWatch, um Statusinformationen für die Anwendung zu verfolgen und in Ihrem Namen KCL-Metriken an CloudWatch diese zu senden. CloudWatch Weitere Informationen zur KCL finden Sie unter Entwickeln Sie KCL 1.x-Verbraucher.

Weitere Informationen zu IAM finden Sie unter:

- AWS Identity and Access Management (IAM)
- Erste Schritte mit IAM
- IAM Benutzerhandbuch

Weitere Informationen zu IAM und Amazon DynamoDB finden Sie unter Verwendung von IAM zur Steuerung des Zugriffs auf Amazon-DynamoDB-Ressourcen im Amazon DynamoDB-Entwickerhandbuch.

Weitere Informationen zu IAM und Amazon CloudWatch finden Sie unter Steuern des Benutzerzugriffs auf Ihr AWS Konto im CloudWatch Amazon-Benutzerhandbuch.

Inhalt

- Richtliniensyntax
- Aktionen für Kinesis Data Streams
- Amazon-Ressourcennamen (ARNs) für Kinesis Data Streams
- Beispielrichtlinien für Kinesis Data Streams
- Teilen Sie Ihren Datenstream mit einem anderen Konto
- Konfigurieren Sie eine AWS Lambda Funktion zum Lesen aus Kinesis Data Streams in einem anderen Konto
- Teilen Sie den Zugriff mithilfe ressourcenbasierter Richtlinien

Richtliniensyntax

Eine IAM-Richtlinie ist ein JSON-Dokument, das eine oder mehrere Anweisungen enthält. Jede Anweisung ist folgendermaßen strukturiert:

```
{
  "Statement":[{
    "Effect":"effect",
    "Action":"action",
    "Resource":"arn",
    "Condition":{
      "condition":{
        "key":"value"
        }
```

```
      }
    }
  ]
}
```

Eine Anweisung kann aus verschiedenen Elementen bestehen:

- Effect: Der effect-Wert kann `Allow` oder `Deny` lauten. IAM-Benutzer verfügen standardmäßig nicht über die Berechtigung zur Verwendung von Ressourcen und API-Aktionen. Daher werden alle Anfragen abgelehnt. Dieser Standardwert kann durch eine explizite Zugriffserlaubnis überschrieben werden. Eine explizite Zugriffsverweigerung überschreibt jedwede Zugriffserlaubnis.
- Action: Mit action wird die API-Aktion spezifiziert, für die Sie Berechtigungen erteilen oder verweigern.
- Resource: Die von einer Aktion betroffene Ressource. Um eine Ressource in der Anweisung anzugeben, benötigen Sie deren Amazon-Ressourcennamen (ARN).
- Condition: Bedingungen sind optional. Mit ihrer Hilfe können Sie bestimmen, wann Ihre Richtlinie wirksam wird.

Beim Erstellen und Verwalten von IAM-Richtlinien sollten Sie den IAM-Richtliniengenerator und den IAM-Richtliniensimulator verwenden.

Aktionen für Kinesis Data Streams

In einer IAM-Richtlinienanweisung können Sie jede API-Aktion von jedem Service, der IAM unterstützt, angeben. Bei Kinesis Data Streams setzen Sie folgendes Präfix vor den Namen der API-Aktion: `kinesis:`. Beispiel: `kinesis:CreateStream`, `kinesis:ListStreams` und `kinesis:DescribeStreamSummary`.

Um mehrere Aktionen in einer einzigen Anweisung anzugeben, trennen Sie sie wie folgt durch Kommata:

```
"Action": ["kinesis:action1", "kinesis:action2"]
```

Sie können auch mehrere Aktionen mittels Platzhaltern angeben. Beispielsweise können Sie alle Aktionen festlegen, deren Name mit dem Wort "Get" beginnt:

```
"Action": "kinesis:Get*"
```

Um alle Operationen von Kinesis Data Streams anzugeben, verwenden Sie den Platzhalter * folgendermaßen:

```
"Action": "kinesis:*"
```

Die vollständige Liste der Kinesis-Data-Streams-API-Aktionen finden Sie unter Amazon-Kinesis-API-Referenz.

Amazon-Ressourcennamen (ARNs) für Kinesis Data Streams

Jede IAM-Richtlinienerklärung gilt für die Ressourcen, die Sie mithilfe ihrer angeben. ARNs

Verwenden Sie das folgende ARN-Ressourcenformat für Kinesis-Datenströme:

```
arn:aws:kinesis:region:account-id:stream/stream-name
```

Beispiel:

```
"Resource": arn:aws:kinesis:*:111122223333:stream/my-stream
```

Beispielrichtlinien für Kinesis Data Streams

Die folgenden Beispielrichtlinien zeigen, wie Sie den Benutzerzugriff auf Kinesis-Datenströme steuern könnten.

Example 1: Allow users to get data from a stream

Example

Diese Richtlinie erlaubt einem Benutzer oder einer Gruppe, die Operationen `DescribeStreamSummary`, `GetShardIterator` und `GetRecords` auf dem angegebenen Stream und `ListStreams` auf einem beliebigen Stream auszuführen. Diese Richtlinie könnte auf Benutzer angewendet werden, die Daten aus einem spezifischen Stream abrufen können sollten.

JSON

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

```
            "Effect": "Allow",
            "Action": [
                "kinesis:Get*",
                "kinesis:DescribeStreamSummary"
            ],
            "Resource": [
            "arn:aws:kinesis:us-east-1:111122223333:stream/stream1"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "kinesis:ListStreams"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

Example 2: Allow users to add data to any stream in the account

Example

Diese Richtlinie erlaubt einem Benutzer oder einer Gruppe, die Operation `PutRecord` mit einem beliebigen Stream des Kontos zu verwenden. Diese Richtlinie könnte auf Benutzer angewendet werden, die Daten zu allen Streams in einem Konto hinzufügen können sollten.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kinesis:PutRecord"
            ],
            "Resource": [
                "arn:aws:kinesis:us-east-1:111122223333:stream/*"
            ]
```

```
        }
    ]
}
```

Example 3: Allow any Kinesis Data Streams action on a specific stream

Example

Diese Richtlinie erlaubt einem Benutzer oder einer Gruppe, eine beliebige Operation von Kinesis Data Streams auf dem angegebenen Stream zu verwenden. Diese Richtlinie könnte auf Benutzer angewendet werden, die administrative Kontrolle über einen bestimmten Stream haben sollten.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": [
            "arn:aws:kinesis:us-east-1:111122223333:stream/stream1"
            ]
        }
    ]
}
```

Example 4: Allow any Kinesis Data Streams action on any stream

Example

Diese Richtlinie erlaubt einem Benutzer oder einer Gruppe, eine beliebige Operation von Kinesis Data Streams auf einem beliebigen Stream in einem Konto zu verwenden. Da diese Richtlinie vollen Zugriff auf alle Ihre Streams gewährt, sollten Sie sie auf Administratoren beschränken.

JSON

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

```
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": [
                "arn:aws:kinesis:*:111122223333:stream/*"
            ]
        }
    ]
}
```

Teilen Sie Ihren Datenstream mit einem anderen Konto

> **Note**
>
> Die Kinesis Producer Library unterstützt derzeit nicht die Angabe eines Stream-ARN beim Schreiben in einen Datenstream. Verwenden Sie das AWS SDK, wenn Sie in einen kontoübergreifenden Datenstream schreiben möchten.

Fügen Sie eine ressourcenbasierte Richtlinie an Ihren Datenstrom an, um einem anderen Konto, einem IAM-Benutzer oder einer IAM-Rolle Zugriff zu gewähren. Ressourcenbasierte Richtlinien sind JSON-Richtliniendokumente, die Sie an eine Ressource, beispielsweise einen Datenstrom, anfügen. Diese Richtlinien erteilen dem angegebenen Prinzipal die Berechtigung zum Ausführen bestimmter Aktionen für diese Ressource und definieren, unter welchen Bedingungen diese gilt. Eine Richtlinie kann mehrere Anweisungen enthalten. Sie müssen in einer ressourcenbasierten Richtlinie einen Prinzipal angeben. Zu den Prinzipalen können Konten, Benutzer, Rollen, Verbundbenutzer oder Dienste gehören. AWS Sie können Richtlinien in der Konsole von Kinesis Data Streams, API oder SDK konfigurieren.

Beachten Sie, dass die Freigabe des Zugriffs für registrierte Verbraucher, wie beispielsweise Enhanced Fan-Out, eine Richtlinie sowohl für den Datenstrom-ARN als auch für den Verbraucher-ARN erfordert.

Aktivieren Sie den kontoübergreifenden Zugriff

Um kontoübergreifenden Zugriff zu ermöglichen, können Sie ein gesamtes Konto oder IAM-Entitäten in einem anderen Konto als Prinzipal in einer ressourcenbasierten Richtlinie angeben. Durch das Hinzufügen eines kontoübergreifenden Auftraggebers zu einer ressourcenbasierten

Richtlinie ist nur die halbe Vertrauensbeziehung eingerichtet. Wenn sich der Prinzipal und die Ressource in getrennten AWS Konten befinden, müssen Sie außerdem eine identitätsbasierte Richtlinie verwenden, um dem Prinzipal Zugriff auf die Ressource zu gewähren. Wenn jedoch eine ressourcenbasierte Richtlinie Zugriff auf einen Prinzipal in demselben Konto gewährt, ist keine zusätzliche identitätsbasierte Richtlinie erforderlich.

Weitere Informationen zur Verwendung ressourcenbasierter Richtlinien für den kontoübergreifenden Zugriff finden Sie unter Kontoübergreifender Ressourcenzugriff in IAM.

Datenstream-Administratoren können mithilfe von AWS Identity and Access Management Richtlinien festlegen, wer Zugriff auf was hat. Das heißt, welcher Prinzipal Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen kann. Das Element `Action` einer JSON-Richtlinie beschreibt die Aktionen, mit denen Sie den Zugriff in einer Richtlinie zulassen oder verweigern können. Richtlinienaktionen haben normalerweise denselben Namen wie der zugehörige AWS API-Vorgang.

Aktionen von Kinesis Data Streams, die freigegeben werden können:

Action	Zugriffsebene
DescribeStreamConsumer	Konsument
DescribeStreamSummary	Datenstrom
GetRecords	Datenstrom
GetShardIterator	Datenstrom
ListShards	Datenstrom
PutRecord	Datenstrom
PutRecords	Datenstrom
SubscribeToShard	Konsument

Im Folgenden finden Sie Beispiele für die Verwendung einer ressourcenbasierten Richtlinie, um kontoübergreifenden Zugriff auf Ihren Datenstrom oder registrierten Verbraucher zu gewähren.

Um eine kontoübergreifende Aktion durchzuführen, müssen Sie den Stream-ARN für den Zugriff auf den Datenstrom und den Verbraucher-ARN für den Zugriff registrierter Verbraucher angeben.

Beispiel für ressourcenbasierte Richtlinien für Kinesis-Datenstreams

Die Freigabe eines registrierten Verbrauchers erfordert aufgrund der erforderlichen Maßnahmen sowohl eine Datenstromrichtlinie als auch eine Verbraucherrichtlinie.

> **Note**
>
> Nachfolgend finden Sie Beispiele für gültige Werte für `Principal`:
>
> - `{"AWS": "123456789012"}`
> - IAM-Benutzer – `{"AWS": "arn:aws:iam::123456789012:user/user-name"}`
> - IAM-Rolle – `{"AWS":["arn:aws:iam::123456789012:role/role-name"]}`
> - Mehrere Prinzipale (kann eine Kombination aus Konto, Benutzer, Rolle sein) – `{"AWS": ["123456789012", "123456789013", "arn:aws:iam::123456789012:user/user-name"]}`

Example 1: Write access to the data stream

Example

JSON

```
{
    "Version":"2012-10-17",
    "Id": "__default_write_policy_ID",
    "Statement": [
        {
            "Sid": "writestatement",
            "Effect": "Allow",
            "Principal": {
                "AWS": "Account12345"
            },
            "Action": [
                "kinesis:DescribeStreamSummary",
                "kinesis:ListShards",
```

```
                "kinesis:PutRecord",
                "kinesis:PutRecords"
            ],
            "Resource": "arn:aws:kinesis:us-east-2:123456789012:stream/
datastreamABC"
        }
    ]
}
```

Example 2: Read access to the data stream

Example

JSON

```
{
    "Version":"2012-10-17",
    "Id": "__default_sharedthroughput_read_policy_ID",
    "Statement": [
        {
            "Sid": "sharedthroughputreadstatement",
            "Effect": "Allow",
            "Principal": {
                "AWS": "Account12345"
            },
            "Action": [
                "kinesis:DescribeStreamSummary",
                "kinesis:ListShards",
                "kinesis:GetRecords",
                "kinesis:GetShardIterator"
            ],
            "Resource": "arn:aws:kinesis:us-east-2:123456789012:stream/
datastreamABC"
        }
    ]
}
```

Example 3: Share enhanced fan-out read access to a registered consumer

Example

Erklärung zur Datenstromrichtlinie:

JSON

```
{
    "Version":"2012-10-17",
    "Id": "__default_sharedthroughput_read_policy_ID",
    "Statement": [
        {
            "Sid": "consumerreadstatement",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:role/role-name"
            },
            "Action": [
                "kinesis:DescribeStreamSummary",
                "kinesis:ListShards"
            ],
            "Resource": "arn:aws:kinesis:us-east-2:123456789012:stream/
datastreamABC"
        }
    ]
}
```

Erklärung zur Verbraucherrichtlinie:

JSON

```
{
    "Version":"2012-10-17",
    "Id": "__default_efo_read_policy_ID",
    "Statement": [
        {
            "Sid": "eforeadstatement",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:role/role-name"
```

```
            },
            "Action": [
                "kinesis:DescribeStreamConsumer",
                "kinesis:SubscribeToShard"
            ],
            "Resource": "arn:aws:kinesis:us-east-2:123456789012:stream/
datastreamABC/consumer/consumerDEF:1674696300"
        }
    ]
}
```

Platzhalter (*) werden für Aktionen oder Prinzipal-Felder nicht unterstützt, um das Prinzip der geringsten Berechtigung aufrecht zu erhalten.

Verwalten Sie die Richtlinie für Ihren Datenstrom programmatisch

Darüber hinaus verfügt Kinesis Data Streams über drei APIS zur Verwaltung Ihrer Datenstream-Richtlinie: AWS-Managementkonsole

- PutResourcePolicy
- GetResourcePolicy
- DeleteResourcePolicy

Verwenden Sie `PutResourePolicy`, um eine Richtlinie für einen Datenstrom oder Verbraucher anzufügen oder zu überschreiben. Verwenden Sie `GetResourcePolicy`, um eine Richtlinie für den angegebenen Datenstrom oder Verbraucher zu überprüfen und anzuzeigen. Verwenden Sie `DeleteResourcePolicy`, um eine Richtlinie für den angegebenen Datenstrom oder Verbraucher zu löschen.

Richtlinienbeschränkungen

Für die Ressourcenrichtlinien von Kinesis Data Streams gelten folgende Einschränkungen:

- Platzhalter (*) werden nicht unterstützt, um zu verhindern, dass umfassender Zugriff über die Ressourcenrichtlinien gewährt wird, die direkt mit einem Datenstrom oder einem registrierten Verbraucher verknüpft sind. Prüfen Sie außerdem sorgfältig die folgenden Richtlinien, um sicherzustellen, dass sie keinen breiten Zugriff gewähren:

  - Identitätsbasierte Richtlinien, die mit zugehörigen AWS Prinzipalen verknüpft sind (z. B. IAM-Rollen)
  - Ressourcenbasierte Richtlinien, die mit zugehörigen AWS Ressourcen verknüpft sind (z. B. KMS-Schlüssel) AWS Key Management Service
- AWS Service Principals werden für Principals nicht unterstützt, um zu verhindern, dass Stellvertreter verwirrt werden.
- Verbund-Prinzipale werden nicht unterstützt.
- Canonical−Benutzer werden nicht unterstützt. IDs
- Die Größe der Richtlinie darf 20 KB nicht überschreiten.

Teilen Sie den Zugriff auf verschlüsselte Daten

Wenn Sie die serverseitige Verschlüsselung für einen Datenstrom mit AWS verwaltetem KMS-Schlüssel aktiviert haben und den Zugriff über eine Ressourcenrichtlinie gemeinsam nutzen möchten, müssen Sie zur Verwendung des vom Kunden verwalteten Schlüssels (CMK) wechseln. Weitere Informationen finden Sie unter Was ist serverseitige Verschlüsselung für Kinesis Data Streams?. Darüber hinaus müssen Sie Ihren freigebenden Prinzipal-Entitäten Zugriff auf Ihren CMK gewähren, indem Sie die kontoübergreifenden KMS-Freigabe-Funktionen nutzen. Stellen Sie sicher, dass Sie auch die IAM-Richtlinien für die freigebenden Prinzipal-Entitäten ändern. Weitere Informationen finden Sie unter Benutzern in anderen Konten die Verwendung eines KMS-Schlüssels erlauben.

Konfigurieren Sie eine AWS Lambda Funktion zum Lesen aus Kinesis Data Streams in einem anderen Konto

Ein Beispiel dafür, wie Sie eine Lambda-Funktion zum Lesen von Kinesis Data Streams in einem anderen Konto konfigurieren können, finden Sie unter Teilen Sie den Zugriff mit kontoübergreifenden Funktionen AWS Lambda.

Teilen Sie den Zugriff mithilfe ressourcenbasierter Richtlinien

> **Note**
>
> Das Aktualisieren einer vorhandenen ressourcenbasierten Richtlinie bedeutet, dass die bestehende Richtlinie ersetzt wird. Stellen Sie daher sicher, dass Sie in Ihrer neuen Richtlinie alle erforderlichen Informationen angeben.

Teilen Sie den Zugriff mit kontoübergreifenden Funktionen AWS Lambda

Lambda-Operator

1. Gehen Sie zur IAM-Konsole, um eine IAM-Rolle zu erstellen, die als Lambda-Ausführungsrolle für Ihre Funktion verwendet wird. AWS Lambda Fügen Sie die verwaltete IAM-Richtlinie hinzu`AWSLambdaKinesisExecutionRole`, die über die erforderlichen Kinesis Data Streams- und Lambda-Aufrufberechtigungen verfügt. Diese Richtlinie gewährt auch Zugriff auf alle potenziellen Ressourcen von Kinesis Data Streams, auf die Sie möglicherweise Zugriff haben.
2. Erstellen Sie in der AWS Lambda Konsole eine AWS Lambda Funktion zur Verarbeitung von Datensätzen in einem Kinesis Data Streams Streams-Datenstream und wählen Sie während der Einrichtung für die Ausführungsrolle die Rolle aus, die Sie im vorherigen Schritt erstellt haben.
3. Stellen Sie dem Ressourcenbesitzer in Kinesis Data Streams die Ausführungsrolle zur Konfiguration der Ressourcenrichtlinie bereit.
4. Schließen Sie die Einrichtung der Lambda-Funktion ab.

Besitzer der Ressource in Kinesis Data Streams

1. Rufen Sie die kontoübergreifende Lambda-Ausführungsrolle ab, welche die Lambda-Funktion aufruft.
2. Wählen Sie in der Konsole von Amazon Kinesis Data Streams den Datenstrom aus. Wählen Sie die Registerkarte Datenstrom-Freigabe und anschließend die Schaltfläche Freigaberichtlinie erstellen, um den visuellen Richtlinieneditor zu starten. Um einen registrierten Verbraucher innerhalb eines Datenstroms freizugeben, wählen Sie den Verbraucher aus und wählen Sie dann Freigaberichtlinie erstellen aus. Sie können die JSON-Richtlinie auch direkt schreiben.
3. Geben Sie die kontoübergreifende Lambda-Ausführungsrolle als Prinzipal und die genauen Aktionen von Kinesis Data Streams an, auf die Sie Zugriff gewähren. Stellen Sie sicher, dass Sie die Aktion `kinesis:DescribeStream` einbeziehen. Weitere Informationen über beispielhafte Ressourcenrichtlinien für Kinesis Data Streams finden Sie unter Beispiel für ressourcenbasierte Richtlinien für Kinesis-Datenstreams.
4. Wählen Sie Richtlinie erstellen oder verwenden Sie die PutResourcePolicy, um die Richtlinie an Ihre Ressource anzuhängen.

Teilen Sie den Zugriff mit kontoübergreifenden KCL-Verbrauchern

- Wenn Sie KCL 1.x verwenden, stellen Sie sicher, dass Sie KCL 1.15.0 oder höher verwenden.

- Wenn Sie KCL 2.x verwenden, stellen Sie sicher, dass Sie KCL 2.5.3 oder höher verwenden.

KCL-Operator

1. Stellen Sie dem Ressourcenbesitzer Ihren IAM-Benutzer oder Ihre IAM-Rolle zur Ausführung der KCL-Anwendung zur Verfügung.
2. Fragen Sie den Ressourcenbesitzer nach dem Datenstrom oder dem Verbraucher-ARN.
3. Stellen Sie sicher, dass Sie den bereitgestellten Stream-ARN als Teil Ihrer KCL-Konfiguration angeben.

   - Für KCL 1.x: Verwenden Sie den KinesisClientLibConfigurationKonstruktor und geben Sie den Stream-ARN an.
   - Für KCL 2.x: Sie können nur den Stream-ARN oder die Kinesis StreamTrackerClient Library bereitstellen. ConfigsBuilder Geben Sie für StreamTracker den Stream-ARN und die Erstellungsepoche aus der DynamoDB-Leasetabelle an, die von der Bibliothek generiert wird. Wenn Sie von einem gemeinsamen registrierten Benutzer lesen möchten, z. B. Enhanced Fan-Out, verwenden Sie den ARN StreamTracker und geben Sie dem Verbraucher auch den ARN an.

Besitzer der Ressource in Kinesis Data Streams

1. Rufen Sie den kontoübergreifenden IAM-Benutzer oder die IAM-Rolle ab, welcher die KCL-Anwendung ausführt.
2. Wählen Sie in der Konsole von Amazon Kinesis Data Streams den Datenstrom aus. Wählen Sie die Registerkarte Datenstrom-Freigabe und anschließend die Schaltfläche Freigaberichtlinie erstellen, um den visuellen Richtlinieneditor zu starten. Um einen registrierten Verbraucher innerhalb eines Datenstroms freizugeben, wählen Sie den Verbraucher aus und wählen Sie dann Freigaberichtlinie erstellen aus. Sie können die JSON-Richtlinie auch direkt schreiben.
3. Geben Sie den IAM-Benutzer oder die IAM-Rolle der kontoübergreifenden KCL-Anwendung als Prinzipal und die genauen Aktionen in Kinesis Data Streams an, auf die Sie Zugriff gewähren. Weitere Informationen über beispielhafte Ressourcenrichtlinien für Kinesis Data Streams finden Sie unter Beispiel für ressourcenbasierte Richtlinien für Kinesis-Datenstreams.
4. Wählen Sie Richtlinie erstellen oder verwenden Sie die PutResourcePolicy, um die Richtlinie an Ihre Ressource anzuhängen.

Teilen Sie den Zugriff auf verschlüsselte Daten

Wenn Sie die serverseitige Verschlüsselung für einen Datenstrom mit AWS verwaltetem KMS-Schlüssel aktiviert haben und den Zugriff über eine Ressourcenrichtlinie gemeinsam nutzen möchten, müssen Sie zur Verwendung des vom Kunden verwalteten Schlüssels (CMK) wechseln. Weitere Informationen finden Sie unter Was ist serverseitige Verschlüsselung für Kinesis Data Streams?. Darüber hinaus müssen Sie Ihren freigebenden Prinzipal-Entitäten Zugriff auf Ihren CMK gewähren, indem Sie die kontoübergreifenden KMS-Freigabe-Funktionen nutzen. Stellen Sie sicher, dass Sie auch die IAM-Richtlinien für die freigebenden Prinzipal-Entitäten ändern. Weitere Informationen finden Sie unter Benutzern in anderen Konten die Verwendung eines KMS-Schlüssels erlauben.

Konformitätsprüfung für Amazon Kinesis Data Streams

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.

Resilienz in Amazon Kinesis Data Streams

Die AWS globale Infrastruktur basiert auf AWS Regionen und 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 Availability Zones ausführen, ohne dass es zu Unterbrechungen kommt. Availability Zones sind besser hoch 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.

Zusätzlich zur AWS globalen Infrastruktur bietet Kinesis Data Streams mehrere Funktionen, die Sie bei der Unterstützung Ihrer Datenausfallsicherheit und Ihrer Backup-Anforderungen unterstützen.

Notfallwiederherstellung in Amazon Kinesis Data Streams

Wenn Sie Daten aus einem Stream mit einer Anwendung von Amazon Kinesis Data Streams verarbeiten, sind Ausfälle auf folgenden Ebenen möglich:

- Ausfall eines Datensatzprozessors
- Ausfall eines Auftragnehmers oder einer Instance der Anwendung, die den Auftragnehmer instanziiert hat
- Ausfall einer EC2 Instance, die eine oder mehrere Instances der Anwendung hostet

Prozessorausfall aufzeichnen

Der Worker ruft mithilfe von ExecutorServiceJava-Aufgaben Methoden der Datensatzverarbeitung auf. Schlägt die Ausführung einer Aufgabe fehlt, behält der Auftragnehmer die Kontrolle über den Shard, den der Datensatzprozessor verarbeitet hat. Der Auftragnehmer startet eine neue Datensatzprozessoraufgabe für die Verarbeitung des Shards. Weitere Informationen finden Sie unter Drosselung von Lesevorgängen.

Fehler beim Worker oder bei der Anwendung

Wenn ein Worker oder eine Instance der Anwendung Amazon Kinesis Data Streams ausfällt, sollten Sie die Situation erkennen können und angemessen reagieren. Löst beispielsweise die `Worker.run`-Methode eine Ausnahme aus, müssen Sie diese abfangen und verwalten.

Fällt die Anwendung selbst aus, sollten Sie dies erkennen, damit Sie einen Neustart durchführen können. Wenn die Anwendung gestartet wird, instanziiert sie einen neuen Auftragnehmer, der wiederum neue Datensatzprozessoren instanziiert, denen automatisch Shards zur Verarbeitung zugewiesen werden. Dies können dieselben Shards sein, die der Datensatzprozessor vor dem Ausfall verarbeitet hat, oder neue Shards.

Falls der Auftragnehmer oder die Anwendung ausfällt, der Ausfall nicht bemerkt wird und noch weitere Instances der Anwendung auf anderen EC2-Instances ausgeführt werden, wird der Ausfall von den Auftragnehmern dieser weiteren Instances verwaltet. Sie erstellen weitere Datensatzprozessoren für die Verarbeitung der Shards des ausgefallenen Auftragnehmers. Die Verarbeitungslast für diese EC2 Instances erhöht sich entsprechend.

Bei dem hier beschriebenen Szenario wird davon ausgegangen, dass die hostende EC2-Instance trotz des Ausfalls des Auftragnehmers oder der Anwendung weiterhin ausgeführt wird und deshalb nicht von einer Auto-Scaling-Gruppe neu gestartet wird.

Fehler Amazon EC2 EC2-Instance

Wir empfehlen die Ausführung der EC2 Instances Ihrer Anwendung in einer Auto-Scaling-Gruppe. Wenn dann eine der EC2 Instances ausfällt, startet die Auto-Scaling-Gruppe automatisch eine neue Instance. Sie sollten die Instances so konfigurieren, dass Ihre Anwendung Amazon Kinesis Data Streams beim Start gestartet wird.

Infrastruktursicherheit in Amazon Kinesis Data Streams

Als verwalteter Service ist er durch 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 für den Zugriff über das Netzwerk. 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.

Bewährte Sicherheitsmethoden für Kinesis Data Streams

Amazon Kinesis Data Streams enthält eine Reihe von Sicherheitsfunktionen, die Sie bei der Entwicklung und Implementierung Ihrer eigenen Sicherheitsrichtlinien berücksichtigen sollten. Die folgenden bewährten Methoden sind allgemeine Richtlinien und keine vollständige Sicherheitslösung. Da diese bewährten Methoden für Ihre Umgebung möglicherweise nicht angemessen oder ausreichend sind, sollten Sie sie als hilfreiche Überlegungen und nicht als bindend ansehen.

Implementieren des Zugriffs mit geringsten Berechtigungen

Beim Erteilen von Berechtigungen entscheiden Sie, wer welche Berechtigungen für welche Ressourcen von Kinesis Data Streams erhält. Sie aktivieren die spezifischen Aktionen, die daraufhin für die betreffenden Ressourcen erlaubt sein sollen. Aus diesem Grund sollten Sie nur Berechtigungen gewähren, die zum Ausführen einer Aufgabe erforderlich sind. Die Implementierung der geringstmöglichen Zugriffsrechte ist eine grundlegende Voraussetzung zum Reduzieren des Sicherheitsrisikos und der Auswirkungen, die aufgrund von Fehlern oder böswilligen Absichten entstehen könnten.

Verwenden von IAM-Rollen

Produzenten- und Client-Anwendungen müssen über gültige Anmeldeinformationen für den Zugriff auf Kinesis-Datenströme verfügen. Sie sollten AWS Anmeldeinformationen nicht direkt in einer Client-Anwendung oder in einem Amazon S3 S3-Bucket speichern. Dabei handelt es sich um langfristige Anmeldeinformationen, die nicht automatisch rotiert werden und bedeutende geschäftliche Auswirkungen haben könnten, wenn sie kompromittiert werden.

Stattdessen sollten Sie eine IAM-Rolle zum Verwalten von temporären Anmeldeinformationen für Ihre Produzenten- und Client-Anwendungen für den Zugriff auf Kinesis-Datenströme verwenden. Wenn Sie eine Rolle verwenden, müssen Sie keine langfristigen Anmeldeinformationen (z. B. Benutzername und Passwort oder Zugriffsschlüssel) für den Zugriff auf andere Ressourcen verwenden.

Weitere Informationen finden Sie unter folgenden Themen im IAM-Benutzerhandbuch:

- IAM-Rollen
- Gängige Szenarien für Rollen: Benutzer, Anwendungen und Services

Implementieren Sie serverseitige Verschlüsselung in abhängigen Ressourcen

Daten im Ruhezustand und Daten während der Übertragung können in Kinesis Data Streams verschlüsselt werden. Weitere Informationen finden Sie unter Datenschutz in Amazon Kinesis Data Streams.

Wird CloudTrail zur Überwachung von API-Aufrufen verwendet

Kinesis Data Streams ist in einen Dienst integriert AWS CloudTrail, der eine Aufzeichnung der Aktionen bereitstellt, die von einem Benutzer, einer Rolle oder einem AWS Dienst in Kinesis Data Streams ausgeführt wurden.

Anhand der von gesammelten Informationen können Sie die Anfrage CloudTrail, die an Kinesis Data Streams gestellt wurde, die IP-Adresse, von der aus die Anfrage gestellt wurde, wer die Anfrage gestellt hat, wann sie gestellt wurde, und weitere Details ermitteln.

Weitere Informationen finden Sie unter the section called “API-Aufrufe von Amazon Kinesis Data Streams protokollieren mit AWS CloudTrail”.

Verwenden Sie diesen Dienst mit einem SDK AWS

AWS Software Development Kits (SDKs) sind für viele beliebte Programmiersprachen verfügbar. Jedes SDK bietet eine API, Codebeispiele und Dokumentation, die es Entwicklern erleichtern, Anwendungen in ihrer bevorzugten Sprache zu erstellen.

SDK-Dokumentation	Codebeispiele
AWS SDK für C++	AWS SDK für C++ Codebeispiele
AWS CLI	AWS CLI Code-Beispiele
AWS SDK für Go	AWS SDK für Go Code-Beispiele
AWS SDK für Java	AWS SDK für Java Code-Beispiele
AWS SDK für JavaScript	AWS SDK für JavaScript Code-Beispiele
AWS SDK für Kotlin	AWS SDK für Kotlin Code-Beispiele
AWS SDK für .NET	AWS SDK für .NET Code-Beispiele
AWS SDK für PHP	AWS SDK für PHP Code-Beispiele
AWS -Tools für PowerShell	AWS -Tools für PowerShell Code-Beispiele
AWS SDK für Python (Boto3)	AWS SDK für Python (Boto3) Code-Beispiele
AWS SDK für Ruby	AWS SDK für Ruby Code-Beispiele
AWS SDK für Rust	AWS SDK für Rust Code-Beispiele
AWS SDK für SAP ABAP	AWS SDK für SAP ABAP Code-Beispiele
AWS SDK für Swift	AWS SDK für Swift Code-Beispiele

Beispiel für die Verfügbarkeit

Sie können nicht finden, was Sie brauchen? Fordern Sie ein Codebeispiel an, indem Sie unten den Link Provide feedback (Feedback geben) auswählen.

Codebeispiele für Kinesis mit AWS SDKs

Die folgenden Codebeispiele zeigen, wie Kinesis mit einem AWS Software Development Kit (SDK) verwendet wird.

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.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Dienst mit einem SDK AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Codebeispiele

- Grundlegende Beispiele für die Verwendung von Kinesis AWS SDKs
  - Erlernen Sie die Grundlagen von Kinesis mit einem SDK AWS
  - Aktionen für Kinesis mit AWS SDKs
    - Verwendung AddTagsToStream mit einem AWS SDK oder CLI
    - Verwendung CreateStream mit einem AWS SDK oder CLI
    - Verwendung DeleteStream mit einem AWS SDK oder CLI
    - Verwendung DeregisterStreamConsumer mit einem AWS SDK oder CLI
    - Verwendung DescribeStream mit einem AWS SDK oder CLI
    - Verwendung GetRecords mit einem AWS SDK oder CLI
    - Verwendung von GetShardIterator mit einer CLI
    - ListStreamConsumersMit einem AWS SDK verwenden
    - Verwendung ListStreams mit einem AWS SDK oder CLI
    - Verwendung ListTagsForStream mit einem AWS SDK oder CLI
    - Verwendung PutRecord mit einem AWS SDK oder CLI
    - Verwendung PutRecords mit einem AWS SDK oder CLI
    - Verwendung RegisterStreamConsumer mit einem AWS SDK oder CLI
- Serverless-Beispiele für Kinesis

- Aufrufen einer Lambda-Funktion über einen Kinesis-Auslöser
- Melden von Batch-Elementfehlern für Lambda-Funktionen mit einem Kinesis-Auslöser

Grundlegende Beispiele für die Verwendung von Kinesis AWS SDKs

Die folgenden Codebeispiele zeigen, wie Sie die Grundlagen von Amazon Kinesis mit AWS SDKs verwenden können.

Beispiele

- Erlernen Sie die Grundlagen von Kinesis mit einem SDK AWS
- Aktionen für Kinesis mit AWS SDKs
  - Verwendung AddTagsToStream mit einem AWS SDK oder CLI
  - Verwendung CreateStream mit einem AWS SDK oder CLI
  - Verwendung DeleteStream mit einem AWS SDK oder CLI
  - Verwendung DeregisterStreamConsumer mit einem AWS SDK oder CLI
  - Verwendung DescribeStream mit einem AWS SDK oder CLI
  - Verwendung GetRecords mit einem AWS SDK oder CLI
  - Verwendung von GetShardIterator mit einer CLI
  - ListStreamConsumersMit einem AWS SDK verwenden
  - Verwendung ListStreams mit einem AWS SDK oder CLI
  - Verwendung ListTagsForStream mit einem AWS SDK oder CLI
  - Verwendung PutRecord mit einem AWS SDK oder CLI
  - Verwendung PutRecords mit einem AWS SDK oder CLI
  - Verwendung RegisterStreamConsumer mit einem AWS SDK oder CLI

Erlernen Sie die Grundlagen von Kinesis mit einem SDK AWS

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Erstellen Sie einen Stream und fügen Sie einen Datensatz ein.
- Erstellen Sie einen Shard-Iterator.

- Lesen Sie den Datensatz und bereinigen Sie anschließend die Ressourcen.

SAP ABAP

SDK für SAP ABAP

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
   DATA lo_stream_describe_result TYPE REF TO /aws1/cl_knsdescrstreamoutput.
   DATA lo_stream_description TYPE REF TO /aws1/cl_knsstreamdescription.
   DATA lo_sharditerator TYPE REF TO /aws1/cl_knsgetsharditerator01.
   DATA lo_record_result TYPE REF TO /aws1/cl_knsputrecordoutput.

   "Create stream."
   TRY.
       lo_kns->createstream(
           iv_streamname = iv_stream_name
           iv_shardcount = iv_shard_count ).
       MESSAGE 'Stream created.' TYPE 'I'.
     CATCH /aws1/cx_knsinvalidargumentex.
       MESSAGE 'The specified argument was not valid.' TYPE 'E'.
     CATCH /aws1/cx_knslimitexceededex.
       MESSAGE 'The request processing has failed because of a limit exceeded
exception.' TYPE 'E'.
     CATCH /aws1/cx_knsresourceinuseex.
       MESSAGE 'The request processing has failed because the resource is in
use.' TYPE 'E'.
   ENDTRY.

   "Wait for stream to becomes active."
   lo_stream_describe_result = lo_kns->describestream( iv_streamname =
iv_stream_name ).
   lo_stream_description = lo_stream_describe_result->get_streamdescription( ).
   WHILE lo_stream_description->get_streamstatus( ) <> 'ACTIVE'.
     IF sy-index = 30.
       EXIT.               "maximum 5 minutes"
     ENDIF.
```

```
      WAIT UP TO 10 SECONDS.
      lo_stream_describe_result = lo_kns->describestream( iv_streamname =
 iv_stream_name ).
      lo_stream_description = lo_stream_describe_result-
>get_streamdescription( ).
    ENDWHILE.

    "Create record."
    TRY.
        lo_record_result = lo_kns->putrecord(
            iv_streamname = iv_stream_name
            iv_data       = iv_data
            iv_partitionkey = iv_partition_key ).
        MESSAGE 'Record created.' TYPE 'I'.
      CATCH /aws1/cx_knsinvalidargumentex.
        MESSAGE 'The specified argument was not valid.' TYPE 'E'.
      CATCH /aws1/cx_knskmsaccessdeniedex.
        MESSAGE 'You do not have permission to perform this AWS KMS action.' TYPE
 'E'.
      CATCH /aws1/cx_knskmsdisabledex.
        MESSAGE 'KMS key used is disabled.' TYPE 'E'.
      CATCH /aws1/cx_knskmsinvalidstateex.
        MESSAGE 'KMS key used is in an invalid state. ' TYPE 'E'.
      CATCH /aws1/cx_knskmsnotfoundex.
        MESSAGE 'KMS key used is not found.' TYPE 'E'.
      CATCH /aws1/cx_knskmsoptinrequired.
        MESSAGE 'KMS key option is required.' TYPE 'E'.
      CATCH /aws1/cx_knskmsthrottlingex.
        MESSAGE 'The rate of requests to AWS KMS is exceeding the request
 quotas.' TYPE 'E'.
      CATCH /aws1/cx_knsprovthruputexcdex.
        MESSAGE 'The request rate for the stream is too high, or the requested
 data is too large for the available throughput.' TYPE 'E'.
      CATCH /aws1/cx_knsresourcenotfoundex.
        MESSAGE 'Resource being accessed is not found.' TYPE 'E'.
    ENDTRY.

    "Create a shard iterator in order to read the record."
    TRY.
        lo_sharditerator = lo_kns->getsharditerator(
          iv_shardid = lo_record_result->get_shardid( )
          iv_sharditeratortype = iv_sharditeratortype
          iv_streamname = iv_stream_name ).
        MESSAGE 'Shard iterator created.' TYPE 'I'.
```

```
     CATCH /aws1/cx_knsinvalidargumentex.
       MESSAGE 'The specified argument was not valid.' TYPE 'E'.
     CATCH /aws1/cx_knsprovthruputexcdex.
       MESSAGE 'The request rate for the stream is too high, or the requested
data is too large for the available throughput.' TYPE 'E'.
     CATCH /aws1/cx_sgmresourcenotfound.
       MESSAGE 'Resource being accessed is not found.' TYPE 'E'.
   ENDTRY.

   "Read the record."
   TRY.
       oo_result = lo_kns->getrecords(                    " oo_result is
returned for testing purposes. "
           iv_sharditerator   = lo_sharditerator->get_sharditerator( ) ).
       MESSAGE 'Shard iterator created.' TYPE 'I'.
     CATCH /aws1/cx_knsexpirediteratorex.
       MESSAGE 'Iterator expired.' TYPE 'E'.
     CATCH /aws1/cx_knsinvalidargumentex.
       MESSAGE 'The specified argument was not valid.' TYPE 'E'.
     CATCH /aws1/cx_knskmsaccessdeniedex.
       MESSAGE 'You do not have permission to perform this AWS KMS action.' TYPE
'E'.
     CATCH /aws1/cx_knskmsdisabledex.
       MESSAGE 'KMS key used is disabled.' TYPE 'E'.
     CATCH /aws1/cx_knskmsinvalidstateex.
       MESSAGE 'KMS key used is in an invalid state. ' TYPE 'E'.
     CATCH /aws1/cx_knskmsnotfoundex.
       MESSAGE 'KMS key used is not found.' TYPE 'E'.
     CATCH /aws1/cx_knskmsoptinrequired.
       MESSAGE 'KMS key option is required.' TYPE 'E'.
     CATCH /aws1/cx_knskmsthrottlingex.
       MESSAGE 'The rate of requests to AWS KMS is exceeding the request
quotas.' TYPE 'E'.
     CATCH /aws1/cx_knsprovthruputexcdex.
       MESSAGE 'The request rate for the stream is too high, or the requested
data is too large for the available throughput.' TYPE 'E'.
     CATCH /aws1/cx_knsresourcenotfoundex.
       MESSAGE 'Resource being accessed is not found.' TYPE 'E'.
   ENDTRY.

   "Delete stream."
   TRY.
       lo_kns->deletestream(
           iv_streamname = iv_stream_name ).
```

```
        MESSAGE 'Stream deleted.' TYPE 'I'.
      CATCH /aws1/cx_knslimitexceededex.
        MESSAGE 'The request processing has failed because of a limit exceeded
exception.' TYPE 'E'.
      CATCH /aws1/cx_knsresourceinuseex.
        MESSAGE 'The request processing has failed because the resource is in
use.' TYPE 'E'.
    ENDTRY.
```

- Weitere API-Informationen finden Sie in den folgenden Themen der API-Referenz zum AWS -SDK für SAP ABAP.
  - CreateStream
  - DeleteStream
  - GetRecords
  - GetShardIterator
  - PutRecord

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Dienst mit einem SDK AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Aktionen für Kinesis mit AWS SDKs

Die folgenden Codebeispiele zeigen, wie Sie einzelne Kinesis-Aktionen mit AWS SDKs ausführen. Jedes Beispiel enthält einen Link zu GitHub, wo Sie Anweisungen zum Einrichten und Ausführen des Codes finden.

Die folgenden Beispiele enthalten nur die am häufigsten verwendeten Aktionen. Eine vollständige Liste finden Sie in der API-Referenz für Amazon Kinesis.

Beispiele

- Verwendung AddTagsToStream mit einem AWS SDK oder CLI
- Verwendung CreateStream mit einem AWS SDK oder CLI
- Verwendung DeleteStream mit einem AWS SDK oder CLI
- Verwendung DeregisterStreamConsumer mit einem AWS SDK oder CLI
- Verwendung DescribeStream mit einem AWS SDK oder CLI

- Verwendung GetRecords mit einem AWS SDK oder CLI
- Verwendung von GetShardIterator mit einer CLI
- ListStreamConsumersMit einem AWS SDK verwenden
- Verwendung ListStreams mit einem AWS SDK oder CLI
- Verwendung ListTagsForStream mit einem AWS SDK oder CLI
- Verwendung PutRecord mit einem AWS SDK oder CLI
- Verwendung PutRecords mit einem AWS SDK oder CLI
- Verwendung RegisterStreamConsumer mit einem AWS SDK oder CLI

Verwendung **`AddTagsToStream`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `AddTagsToStream` verwendet wird.

.NET

SDK für .NET

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.Kinesis;
using Amazon.Kinesis.Model;

/// <summary>
/// This example shows how to apply key/value pairs to an Amazon Kinesis
/// stream.
/// </summary>
public class TagStream
{
    public static async Task Main()
    {
        IAmazonKinesis client = new AmazonKinesisClient();
```

```
            string streamName = "AmazonKinesisStream";
            var tags = new Dictionary<string, string>
            {
                { "Project", "Sample Kinesis Project" },
                { "Application", "Sample Kinesis App" },
            };

            var success = await ApplyTagsToStreamAsync(client, streamName, tags);

            if (success)
            {
                Console.WriteLine($"Taggs successfully added to {streamName}.");
            }
            else
            {
                Console.WriteLine("Tags were not added to the stream.");
            }
        }

        /// <summary>
        /// Applies the set of tags to the named Kinesis stream.
        /// </summary>
        /// <param name="client">The initialized Kinesis client.</param>
        /// <param name="streamName">The name of the Kinesis stream to which
        /// the tags will be attached.</param>
        /// <param name="tags">A sictionary containing key/value pairs which
        /// will be used to create the Kinesis tags.</param>
        /// <returns>A Boolean value which represents the success or failure
        /// of AddTagsToStreamAsync.</returns>
        public static async Task<bool> ApplyTagsToStreamAsync(
            IAmazonKinesis client,
            string streamName,
            Dictionary<string, string> tags)
        {
            var request = new AddTagsToStreamRequest
            {
                StreamName = streamName,
                Tags = tags,
            };

            var response = await client.AddTagsToStreamAsync(request);

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
```

```
        }
    }
```

- Einzelheiten zur API finden Sie AddTagsToStreamin der AWS SDK für .NET API-Referenz.

CLI

AWS CLI

So fügen Sie einem Datenstrom Tags hinzu

Im folgenden Beispiel für `add-tags-to-stream` wird dem angegebenen Stream ein Tag mit dem Schlüssel `samplekey` und dem Wert `example` zugewiesen.

```
aws kinesis add-tags-to-stream \
    --stream-name samplestream \
    --tags samplekey=example
```

Mit diesem Befehl wird keine Ausgabe zurückgegeben.

Weitere Informationen finden Sie unter Markieren Ihrer Streams im Entwicklerhandbuch für Amazon Kinesis Data Streams.

- Einzelheiten zur API finden Sie AddTagsToStreamin der AWS CLI Befehlsreferenz.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Dienst mit einem SDK AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`CreateStream`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `CreateStream` verwendet wird.

Beispiele für Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Im folgenden Codebeispiel können Sie diese Aktion im Kontext sehen:

- Kennenlernen der Grundlagen

.NET

SDK für .NET

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
using System;
using System.Threading.Tasks;
using Amazon.Kinesis;
using Amazon.Kinesis.Model;

/// <summary>
/// This example shows how to create a new Amazon Kinesis stream.
/// </summary>
public class CreateStream
{
    public static async Task Main()
    {
        IAmazonKinesis client = new AmazonKinesisClient();

        string streamName = "AmazonKinesisStream";
        int shardCount = 1;

        var success = await CreateNewStreamAsync(client, streamName,
shardCount);
        if (success)
        {
            Console.WriteLine($"The stream, {streamName} successfully
created.");
        }
    }

    /// <summary>
    /// Creates a new Kinesis stream.
    /// </summary>
    /// <param name="client">An initialized Kinesis client.</param>
    /// <param name="streamName">The name for the new stream.</param>
    /// <param name="shardCount">The number of shards the new stream will
```

```
        /// use. The throughput of the stream is a function of the number of
        /// shards; more shards are required for greater provisioned
        /// throughput.</param>
        /// <returns>A Boolean value indicating whether the stream was created.</
returns>
        public static async Task<bool> CreateNewStreamAsync(IAmazonKinesis
 client, string streamName, int shardCount)
        {
            var request = new CreateStreamRequest
            {
                StreamName = streamName,
                ShardCount = shardCount,
            };

            var response = await client.CreateStreamAsync(request);

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
    }
```

- Einzelheiten zur API finden Sie CreateStreamin der AWS SDK für .NET API-Referenz.

CLI

AWS CLI

So erstellen Sie einen Datenstrom

Im folgenden Beispiel für `create-stream` wird ein Datenstrom namens samplestream mit 3 Shards erstellt.

```
aws kinesis create-stream \
    --stream-name samplestream \
    --shard-count 3
```

Mit diesem Befehl wird keine Ausgabe zurückgegeben.

Weitere Informationen finden Sie unter Erstellen eines Streams im Entwicklerhandbuch für Amazon Kinesis Data Streams.

- Einzelheiten zur API finden Sie CreateStreamin der AWS CLI Befehlsreferenz.

Java

SDK für Java 2.x

> **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 software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.kinesis.KinesisClient;
import software.amazon.awssdk.services.kinesis.model.CreateStreamRequest;
import software.amazon.awssdk.services.kinesis.model.KinesisException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class CreateDataStream {
    public static void main(String[] args) {

        final String usage = """

                Usage:
                    <streamName>

                Where:
                    streamName - The Amazon Kinesis data stream (for example,
 StockTradeStream).
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }
```

```
        String streamName = args[0];
        Region region = Region.US_EAST_1;
        KinesisClient kinesisClient = KinesisClient.builder()
                .region(region)
                .build();
        createStream(kinesisClient, streamName);
        System.out.println("Done");
        kinesisClient.close();
    }

    public static void createStream(KinesisClient kinesisClient, String
 streamName) {
        try {
            CreateStreamRequest streamReq = CreateStreamRequest.builder()
                    .streamName(streamName)
                    .shardCount(1)
                    .build();

            kinesisClient.createStream(streamReq);

        } catch (KinesisException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- Einzelheiten zur API finden Sie CreateStreamin der AWS SDK for Java 2.x API-Referenz.

PowerShell

Tools für PowerShell V4

Beispiel 1: Erzeugt einen neuen Stream. Standardmäßig gibt dieses Cmdlet keine Ausgabe zurück, sodass der PassThru Schalter - hinzugefügt wird, um den Wert zurückzugeben, der dem StreamName Parameter - zur späteren Verwendung übergeben wurde.

```
$streamName = New-KINStream -StreamName "mystream" -ShardCount 1 -PassThru
```

- Einzelheiten zur API finden Sie unter CreateStream AWS -Tools für PowerShellCmdlet-Referenz (V4).

Tools für V5 PowerShell

Beispiel 1: Erzeugt einen neuen Stream.

```
New-KINStream -StreamName "mystream" -ShardCount 1
```

- Einzelheiten zur API finden Sie unter CreateStream AWS -Tools für PowerShellCmdlet-Referenz (V5).

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class KinesisStream:
    """Encapsulates a Kinesis stream."""

    def __init__(self, kinesis_client):
        """
        :param kinesis_client: A Boto3 Kinesis client.
        """
        self.kinesis_client = kinesis_client
        self.name = None
        self.details = None
        self.stream_exists_waiter = kinesis_client.get_waiter("stream_exists")


    def create(self, name, wait_until_exists=True):
        """
        Creates a stream.

        :param name: The name of the stream.
        :param wait_until_exists: When True, waits until the service reports that
                                  the stream exists, then queries for its
 metadata.
        """
```

```
        try:
            self.kinesis_client.create_stream(StreamName=name, ShardCount=1)
            self.name = name
            logger.info("Created stream %s.", name)
            if wait_until_exists:
                logger.info("Waiting until exists.")
                self.stream_exists_waiter.wait(StreamName=name)
                self.describe(name)
        except ClientError:
            logger.exception("Couldn't create stream %s.", name)
            raise
```

- Einzelheiten zur API finden Sie CreateStreamin AWS SDK for Python (Boto3) API Reference.

Rust

SDK für Rust

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
async fn make_stream(client: &Client, stream: &str) -> Result<(), Error> {
    client
        .create_stream()
        .stream_name(stream)
        .shard_count(4)
        .send()
        .await?;

    println!("Created stream");

    Ok(())
}
```

- Einzelheiten zur API finden Sie CreateStreamin der API-Referenz zum AWS SDK für Rust.

SAP ABAP

SDK für SAP ABAP

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
   TRY.
       lo_kns->createstream(
           iv_streamname = iv_stream_name
           iv_shardcount = iv_shard_count ).
       MESSAGE 'Stream created.' TYPE 'I'.
     CATCH /aws1/cx_knsinvalidargumentex.
       MESSAGE 'The specified argument was not valid.' TYPE 'E'.
     CATCH /aws1/cx_knslimitexceededex.
       MESSAGE 'The request processing has failed because of a limit exceed
exception.' TYPE 'E'.
     CATCH /aws1/cx_knsresourceinuseex.
       MESSAGE 'The request processing has failed because the resource is in
use.' TYPE 'E'.
   ENDTRY.
```

- Einzelheiten zur API finden Sie CreateStreamin der API-Referenz zum AWS SDK für SAP ABAP.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Dienst mit einem SDK AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`DeleteStream`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `DeleteStream` verwendet wird.

Beispiele für Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Im folgenden Codebeispiel können Sie diese Aktion im Kontext sehen:

- Kennenlernen der Grundlagen

.NET

SDK für .NET

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
using System;
using System.Threading.Tasks;
using Amazon.Kinesis;
using Amazon.Kinesis.Model;

/// <summary>
/// Shows how to delete an Amazon Kinesis stream.
/// </summary>
public class DeleteStream
{
    public static async Task Main()
    {
        IAmazonKinesis client = new AmazonKinesisClient();
        string streamName = "AmazonKinesisStream";

        var success = await DeleteStreamAsync(client, streamName);

        if (success)
        {
            Console.WriteLine($"Stream, {streamName} successfully deleted.");
        }
        else
        {
            Console.WriteLine("Stream not deleted.");
        }
    }
```

```
        /// <summary>
        /// Deletes a Kinesis stream.
        /// </summary>
        /// <param name="client">An initialized Kinesis client object.</param>
        /// <param name="streamName">The name of the string to delete.</param>
        /// <returns>A Boolean value representing the success of the operation.</
returns>
        public static async Task<bool> DeleteStreamAsync(IAmazonKinesis client,
 string streamName)
        {
            // If EnforceConsumerDeletion is true, any consumers
            // of this stream will also be deleted. If it is set
            // to false and this stream has any consumers, the
            // call will fail with a ResourceInUseException.
            var request = new DeleteStreamRequest
            {
                StreamName = streamName,
                EnforceConsumerDeletion = true,
            };

            var response = await client.DeleteStreamAsync(request);

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
    }
```

- Einzelheiten zur API finden Sie DeleteStreamin der AWS SDK für .NET API-Referenz.

CLI

AWS CLI

So löschen Sie einen Datenstrom

Im folgenden Beispiel für `delete-stream` wird der angegebene Datenstrom gelöscht.

```
aws kinesis delete-stream \
    --stream-name samplestream
```

Mit diesem Befehl wird keine Ausgabe zurückgegeben.

Weitere Informationen finden Sie unter Löschen eines Streams im Entwicklerhandbuch für Amazon Kinesis Data Streams.

- Einzelheiten zur API finden Sie DeleteStreamin der AWS CLI Befehlsreferenz.

Java

SDK für Java 2.x

> **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 software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.kinesis.KinesisClient;
import software.amazon.awssdk.services.kinesis.model.DeleteStreamRequest;
import software.amazon.awssdk.services.kinesis.model.KinesisException;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DeleteDataStream {

    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <streamName>

                Where:
                    streamName - The Amazon Kinesis data stream (for example,
 StockTradeStream)
                """;
```

```
        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String streamName = args[0];
        Region region = Region.US_EAST_1;
        KinesisClient kinesisClient = KinesisClient.builder()
                .region(region)
                .build();

        deleteStream(kinesisClient, streamName);
        kinesisClient.close();
        System.out.println("Done");
    }

    public static void deleteStream(KinesisClient kinesisClient, String
 streamName) {
        try {
            DeleteStreamRequest delStream = DeleteStreamRequest.builder()
                    .streamName(streamName)
                    .build();

            kinesisClient.deleteStream(delStream);

        } catch (KinesisException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
    }
}
```

- Einzelheiten zur API finden Sie DeleteStreamin der AWS SDK for Java 2.x API-Referenz.

PowerShell

Tools für PowerShell V4

Beispiel 1: Löscht den angegebenen Stream. Sie werden um eine Bestätigung gebeten, bevor der Befehl ausgeführt wird. Verwenden Sie den Schalter -Force, um die Bestätigungsaufforderung zu unterdrücken.

```
Remove-KINStream -StreamName "mystream"
```

- Einzelheiten zur API finden Sie unter DeleteStream AWS -Tools für PowerShellCmdlet-Referenz (V4).

Tools für V5 PowerShell

Beispiel 1: Löscht den angegebenen Stream. Sie werden um eine Bestätigung gebeten, bevor der Befehl ausgeführt wird. Verwenden Sie den Schalter -Force, um die Bestätigungsaufforderung zu unterdrücken.

```
Remove-KINStream -StreamName "mystream"
```

- Einzelheiten zur API finden Sie unter DeleteStream AWS -Tools für PowerShellCmdlet-Referenz (V5).

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class KinesisStream:
    """Encapsulates a Kinesis stream."""

    def __init__(self, kinesis_client):
        """
        :param kinesis_client: A Boto3 Kinesis client.
        """
        self.kinesis_client = kinesis_client
        self.name = None
        self.details = None
        self.stream_exists_waiter = kinesis_client.get_waiter("stream_exists")


    def delete(self):
```

```
        """
        Deletes a stream.
        """
        try:
            self.kinesis_client.delete_stream(StreamName=self.name)
            self._clear()
            logger.info("Deleted stream %s.", self.name)
        except ClientError:
            logger.exception("Couldn't delete stream %s.", self.name)
            raise
```

- Einzelheiten zur API finden Sie DeleteStreamin AWS SDK for Python (Boto3) API Reference.

Rust

SDK für Rust

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
async fn remove_stream(client: &Client, stream: &str) -> Result<(), Error> {
    client.delete_stream().stream_name(stream).send().await?;

    println!("Deleted stream.");

    Ok(())
}
```

- Einzelheiten zur API finden Sie DeleteStreamin der API-Referenz zum AWS SDK für Rust.

SAP ABAP

SDK für SAP ABAP

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
   TRY.
       lo_kns->deletestream(
           iv_streamname = iv_stream_name ).
       MESSAGE 'Stream deleted.' TYPE 'I'.
     CATCH /aws1/cx_knslimitexceededex.
       MESSAGE 'The request processing has failed because of a limit exceed
exception.' TYPE 'E'.
     CATCH /aws1/cx_knsresourceinuseex.
       MESSAGE 'The request processing has failed because the resource is in
use.' TYPE 'E'.
   ENDTRY.
```

- Einzelheiten zur API finden Sie DeleteStreamin der API-Referenz zum AWS SDK für SAP ABAP.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Dienst mit einem SDK AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`DeregisterStreamConsumer`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `DeregisterStreamConsumer` verwendet wird.

.NET

SDK für .NET

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    using System;
    using System.Threading.Tasks;
    using Amazon.Kinesis;
    using Amazon.Kinesis.Model;

    /// <summary>
    /// Shows how to deregister a consumer from an Amazon Kinesis stream.
    /// </summary>
    public class DeregisterConsumer
    {
        public static async Task Main(string[] args)
        {
            IAmazonKinesis client = new AmazonKinesisClient();

            string streamARN = "arn:aws:kinesis:us-west-2:000000000000:stream/
AmazonKinesisStream";
            string consumerName = "CONSUMER_NAME";
            string consumerARN = "arn:aws:kinesis:us-west-2:000000000000:stream/
AmazonKinesisStream/consumer/CONSUMER_NAME:000000000000";

            var success = await DeregisterConsumerAsync(client, streamARN,
 consumerARN, consumerName);

            if (success)
            {
                Console.WriteLine($"{consumerName} successfully deregistered.");
            }
            else
            {
                Console.WriteLine($"{consumerName} was not successfully
 deregistered.");
            }
```

```
        }

        /// <summary>
        /// Deregisters a consumer from a Kinesis stream.
        /// </summary>
        /// <param name="client">An initialized Kinesis client object.</param>
        /// <param name="streamARN">The ARN of a Kinesis stream.</param>
        /// <param name="consumerARN">The ARN of the consumer.</param>
        /// <param name="consumerName">The name of the consumer.</param>
        /// <returns>A Boolean value representing the success of the operation.</
returns>
        public static async Task<bool> DeregisterConsumerAsync(
            IAmazonKinesis client,
            string streamARN,
            string consumerARN,
            string consumerName)
        {
            var request = new DeregisterStreamConsumerRequest
            {
                StreamARN = streamARN,
                ConsumerARN = consumerARN,
                ConsumerName = consumerName,
            };

            var response = await client.DeregisterStreamConsumerAsync(request);

            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
    }
```

- Einzelheiten zur API finden Sie DeregisterStreamConsumerin der AWS SDK für .NET API-Referenz.

CLI

AWS CLI

So heben Sie die Registrierung eines Datenstromkonsumenten auf

Im folgenden Beispiel für `deregister-stream-consumer` wird die Registrierung des angegebenen Konsumenten beim angegebenen Datenstrom aufgehoben.

```
aws kinesis deregister-stream-consumer \
    --stream-arn arn:aws:kinesis:us-west-2:123456789012:stream/samplestream \
    --consumer-name KinesisConsumerApplication
```

Mit diesem Befehl wird keine Ausgabe zurückgegeben.

Weitere Informationen finden Sie unter Entwickeln von Konsumenten mit Enhanced Fan-Out über die API von Kinesis Data Streams im Entwicklerhandbuch für Amazon Kinesis Data Streams.

- Einzelheiten zur API finden Sie DeregisterStreamConsumerin der AWS CLI Befehlsreferenz.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Dienst mit einem SDK AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`DescribeStream`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `DescribeStream` verwendet wird.

CLI

AWS CLI

So beschreiben Sie einen Datenstrom

Im folgenden Beispiel für `describe-stream` werden Details zum angegebenen Datenstrom abgerufen.

```
aws kinesis describe-stream \
    --stream-name samplestream
```

Ausgabe:

```
{
    "StreamDescription": {
        "Shards": [
            {
                "ShardId": "shardId-000000000000",
                "HashKeyRange": {
                    "StartingHashKey": "0",
                    "EndingHashKey": "113427455640312821154458202477256070484"
```

```
                },
                "SequenceNumberRange": {
                    "StartingSequenceNumber":
 "496008716829570364423650249261910734372510605801286533l4"
                }
            },
            {
                "ShardId": "shardId-000000000001",
                "HashKeyRange": {
                    "StartingHashKey": "113427455640312821154458202477256070485",
                    "EndingHashKey": "226854911280625642308916404954512140969"
                },
                "SequenceNumberRange": {
                    "StartingSequenceNumber":
 "496008716829793371875635555493326091555237089416346337l46"
                }
            },
            {
                "ShardId": "shardId-000000000002",
                "HashKeyRange": {
                    "StartingHashKey": "226854911280625642308916404954512140970",
                    "EndingHashKey": "340282366920938463463374607431768211455"
                },
                "SequenceNumberRange": {
                    "StartingSequenceNumber":
 "496008716830016379327620861724741448737963573031406141l8"
                }
            }
        ],
        "StreamARN": "arn:aws:kinesis:us-west-2:123456789012:stream/
samplestream",
        "StreamName": "samplestream",
        "StreamStatus": "ACTIVE",
        "RetentionPeriodHours": 24,
        "EnhancedMonitoring": [
            {
                "ShardLevelMetrics": []
            }
        ],
        "EncryptionType": "NONE",
        "KeyId": null,
        "StreamCreationTimestamp": 1572297168.0
    }
```

```
}
```

Weitere Informationen finden Sie unter Erstellen und Verwalten von Streams im Entwicklerhandbuch für Amazon Kinesis Data Streams.

- Einzelheiten zur API finden Sie DescribeStreamin der AWS CLI Befehlsreferenz.

PowerShell

Tools für PowerShell V4

Beispiel 1: Gibt Details des angegebenen Streams zurück.

```
Get-KINStream -StreamName "mystream"
```

Ausgabe:

```
HasMoreShards        : False
RetentionPeriodHours : 24
Shards               : {}
StreamARN            : arn:aws:kinesis:us-west-2:123456789012:stream/mystream
StreamName           : mystream
StreamStatus         : ACTIVE
```

- Einzelheiten zur API finden Sie unter DescribeStream AWS -Tools für PowerShellCmdlet-Referenz (V4).

Tools für V5 PowerShell

Beispiel 1: Gibt Details des angegebenen Streams zurück.

```
Get-KINStream -StreamName "mystream"
```

Ausgabe:

```
HasMoreShards        : False
RetentionPeriodHours : 24
Shards               : {}
StreamARN            : arn:aws:kinesis:us-west-2:123456789012:stream/mystream
StreamName           : mystream
```

```
StreamStatus         : ACTIVE
```

- Einzelheiten zur API finden Sie unter DescribeStream AWS -Tools für PowerShellCmdlet-Referenz (V5).

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class KinesisStream:
    """Encapsulates a Kinesis stream."""

    def __init__(self, kinesis_client):
        """
        :param kinesis_client: A Boto3 Kinesis client.
        """
        self.kinesis_client = kinesis_client
        self.name = None
        self.details = None
        self.stream_exists_waiter = kinesis_client.get_waiter("stream_exists")


    def describe(self, name):
        """
        Gets metadata about a stream.

        :param name: The name of the stream.
        :return: Metadata about the stream.
        """
        try:
            response = self.kinesis_client.describe_stream(StreamName=name)
            self.name = name
            self.details = response["StreamDescription"]
            logger.info("Got stream %s.", name)
        except ClientError:
```

```
            logger.exception("Couldn't get %s.", name)
            raise
        else:
            return self.details
```

- Einzelheiten zur API finden Sie DescribeStreamin AWS SDK for Python (Boto3) API Reference.

Rust

SDK für Rust

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
async fn show_stream(client: &Client, stream: &str) -> Result<(), Error> {
    let resp = client.describe_stream().stream_name(stream).send().await?;

    let desc = resp.stream_description.unwrap();

    println!("Stream description:");
    println!("  Name:              {}:", desc.stream_name());
    println!("  Status:            {:?}", desc.stream_status());
    println!("  Open shards:       {:?}", desc.shards.len());
    println!("  Retention (hours): {}", desc.retention_period_hours());
    println!("  Encryption:        {:?}", desc.encryption_type.unwrap());

    Ok(())
}
```

- Einzelheiten zur API finden Sie DescribeStreamin der API-Referenz zum AWS SDK für Rust.

SAP ABAP

SDK für SAP ABAP

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    TRY.
        oo_result = lo_kns->describestream(
            iv_streamname = iv_stream_name ).
        DATA(lt_stream_description) = oo_result->get_streamdescription( ).
        MESSAGE 'Streams retrieved.' TYPE 'I'.
      CATCH /aws1/cx_knslimitexceededex.
        MESSAGE 'The request processing has failed because of a limit exceed
exception.' TYPE 'E'.
      CATCH /aws1/cx_knsresourcenotfoundex.
        MESSAGE 'Resource being accessed is not found.' TYPE 'E'.
    ENDTRY.
```

- Einzelheiten zur API finden Sie DescribeStreamin der API-Referenz zum AWS SDK für SAP ABAP.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Dienst mit einem SDK AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`GetRecords`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `GetRecords` verwendet wird.

Beispiele für Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Im folgenden Codebeispiel können Sie diese Aktion im Kontext sehen:

- Kennenlernen der Grundlagen

CLI

AWS CLI

So rufen Sie Datensätze aus einem Shard ab

Im folgenden Beispiel für `get-records` werden Datensätze aus dem Shard eines Kinesis-Datenstroms über den angegebenen Shard-Iterator abgerufen.

```
aws kinesis get-records \
    --shard-iterator AAAAAAAAAAF7/0mWD7IuHj1yGv/
TKuNgx2ukD5xipCY4cy4gU96orWwZwcSXh3K9tAmGYeOZyLZrvzzeOFVf9iN99hUPw/w/
b0YWYeehfNvnf1DYt5XpDJghLKr3DzgznkTmMymDP3R+3wRKeuEw6/kdxY2yKJH0veaiekaVc4N2VwK/
GvaGP2Hh9Fg7N++q0Adg6fIDQPt4p8RpavDbk+A4sL9SWGE1
```

Ausgabe:

```
{
    "Records": [],
    "MillisBehindLatest": 80742000
}
```

Weitere Informationen finden Sie unter Developing Consumer Using the Kinesis Data Streams API with the AWS SDK for Java im Amazon Kinesis Data Streams Developer Guide.

- Einzelheiten zur API finden Sie GetRecordsin der AWS CLI Befehlsreferenz.

Java

SDK für Java 2.x

> **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 software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.kinesis.KinesisClient;
import software.amazon.awssdk.services.kinesis.model.DescribeStreamResponse;
```

```
import software.amazon.awssdk.services.kinesis.model.DescribeStreamRequest;
import software.amazon.awssdk.services.kinesis.model.Shard;
import software.amazon.awssdk.services.kinesis.model.GetShardIteratorRequest;
import software.amazon.awssdk.services.kinesis.model.GetShardIteratorResponse;
import software.amazon.awssdk.services.kinesis.model.Record;
import software.amazon.awssdk.services.kinesis.model.GetRecordsRequest;
import software.amazon.awssdk.services.kinesis.model.GetRecordsResponse;
import java.util.ArrayList;
import java.util.List;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class GetRecords {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <streamName>

                Where:
                    streamName - The Amazon Kinesis data stream to read from (for
 example, StockTradeStream).
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String streamName = args[0];
        Region region = Region.US_EAST_1;
        KinesisClient kinesisClient = KinesisClient.builder()
                .region(region)
                .build();

        getStockTrades(kinesisClient, streamName);
        kinesisClient.close();
```

```
    }

    public static void getStockTrades(KinesisClient kinesisClient, String
streamName) {
        String shardIterator;
        String lastShardId = null;
        DescribeStreamRequest describeStreamRequest =
DescribeStreamRequest.builder()
                .streamName(streamName)
                .build();

        List<Shard> shards = new ArrayList<>();
        DescribeStreamResponse streamRes;
        do {
            streamRes = kinesisClient.describeStream(describeStreamRequest);
            shards.addAll(streamRes.streamDescription().shards());

            if (shards.size() > 0) {
                lastShardId = shards.get(shards.size() - 1).shardId();
            }
        } while (streamRes.streamDescription().hasMoreShards());

        GetShardIteratorRequest itReq = GetShardIteratorRequest.builder()
                .streamName(streamName)
                .shardIteratorType("TRIM_HORIZON")
                .shardId(lastShardId)
                .build();

        GetShardIteratorResponse shardIteratorResult =
kinesisClient.getShardIterator(itReq);
        shardIterator = shardIteratorResult.shardIterator();

        // Continuously read data records from shard.
        List<Record> records;

        // Create new GetRecordsRequest with existing shardIterator.
        // Set maximum records to return to 1000.
        GetRecordsRequest recordsRequest = GetRecordsRequest.builder()
                .shardIterator(shardIterator)
                .limit(1000)
                .build();

        GetRecordsResponse result = kinesisClient.getRecords(recordsRequest);
```

```
        // Put result into record list. Result may be empty.
        records = result.records();

        // Print records
        for (Record record : records) {
            SdkBytes byteBuffer = record.data();
            System.out.printf("Seq No: %s - %s%n", record.sequenceNumber(), new
 String(byteBuffer.asByteArray()));
        }
    }
}
```

- Einzelheiten zur API finden Sie GetRecordsin der AWS SDK for Java 2.x API-Referenz.

PowerShell

Tools für PowerShell V4

Beispiel 1: In diesem Beispiel wird gezeigt, wie Daten aus einer Reihe von einem oder mehreren Datensätzen zurückgegeben und extrahiert werden können. Der Iterator, an den geliefert Get-KINRecord wird, bestimmt die Startposition der zurückzugebenden Datensätze, die in diesem Beispiel in einer Variablen, $records, erfasst werden. Auf jeden einzelnen Datensatz kann dann durch Indizierung der $records-Sammlung zugegriffen werden. Unter der Annahme, dass es sich bei den Daten im Datensatz um UTF-8-codierten Text handelt, zeigt der letzte Befehl, wie Sie die Daten aus dem Objekt extrahieren und als Text MemoryStream an die Konsole zurückgeben können.

```
$records
$records = Get-KINRecord -ShardIterator "AAAAAAAAAAGIc....9VnbiRNaP"
```

Ausgabe:

```
MillisBehindLatest NextShardIterator           Records
------------------ -----------------           -------
0                  AAAAAAAAAAERNIq...uDn11HuUs {Key1, Key2}
```

```
$records.Records[0]
```

Ausgabe:

```
ApproximateArrivalTimestamp Data                    PartitionKey SequenceNumber
--------------------------- ----                    ------------ --------------
3/7/2016 5:14:33 PM         System.IO.MemoryStream Key1
 4955986459776...931586
```

```
[Text.Encoding]::UTF8.GetString($records.Records[0].Data.ToArray())
```

Ausgabe:

```
test data from string
```

- Einzelheiten zur API finden Sie unter GetRecords AWS -Tools für PowerShellCmdlet-Referenz (V4).

Tools für V5 PowerShell

Beispiel 1: In diesem Beispiel wird gezeigt, wie Daten aus einer Reihe von einem oder mehreren Datensätzen zurückgegeben und extrahiert werden können. Der Iterator, an den geliefert Get-KINRecord wird, bestimmt die Startposition der zurückzugebenden Datensätze, die in diesem Beispiel in einer Variablen, $records, erfasst werden. Auf jeden einzelnen Datensatz kann dann durch Indizierung der $records-Sammlung zugegriffen werden. Unter der Annahme, dass es sich bei den Daten im Datensatz um UTF-8-codierten Text handelt, zeigt der letzte Befehl, wie Sie die Daten aus dem Objekt extrahieren und als Text MemoryStream an die Konsole zurückgeben können.

```
$records
$records = Get-KINRecord -ShardIterator "AAAAAAAAAAGIc....9VnbiRNaP"
```

Ausgabe:

```
MillisBehindLatest NextShardIterator           Records
------------------ -----------------           -------
0                  AAAAAAAAAAERNIq...uDn11HuUs  {Key1, Key2}
```

```
$records.Records[0]
```

Ausgabe:

```
ApproximateArrivalTimestamp Data                    PartitionKey SequenceNumber
```

```
-------------------------- ----                    ------------ --------------
3/7/2016 5:14:33 PM        System.IO.MemoryStream Key1
 4955986459776...931586
```

```
[Text.Encoding]::UTF8.GetString($records.Records[0].Data.ToArray())
```

Ausgabe:

```
test data from string
```

- Einzelheiten zur API finden Sie unter GetRecords AWS -Tools für PowerShellCmdlet-Referenz (V5).

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class KinesisStream:
    """Encapsulates a Kinesis stream."""

    def __init__(self, kinesis_client):
        """
        :param kinesis_client: A Boto3 Kinesis client.
        """
        self.kinesis_client = kinesis_client
        self.name = None
        self.details = None
        self.stream_exists_waiter = kinesis_client.get_waiter("stream_exists")


    def get_records(self, max_records):
        """
        Gets records from the stream. This function is a generator that first
 gets
```

```
        a shard iterator for the stream, then uses the shard iterator to get
records
        in batches from the stream. The shard iterator can be accessed through
the
        'details' property, which is populated using the 'describe' function of
this class.
        Each batch of records is yielded back to the caller until the specified
        maximum number of records has been retrieved.

        :param max_records: The maximum number of records to retrieve.
        :return: Yields the current batch of retrieved records.
        """
        try:
            response = self.kinesis_client.get_shard_iterator(
                StreamName=self.name,
                ShardId=self.details["Shards"][0]["ShardId"],
                ShardIteratorType="LATEST",
            )
            shard_iter = response["ShardIterator"]
            record_count = 0
            while record_count < max_records:
                response = self.kinesis_client.get_records(
                    ShardIterator=shard_iter, Limit=10
                )
                shard_iter = response["NextShardIterator"]
                records = response["Records"]
                logger.info("Got %s records.", len(records))
                record_count += len(records)
                yield records
        except ClientError:
            logger.exception("Couldn't get records from stream %s.", self.name)
            raise



    def describe(self, name):
        """
        Gets metadata about a stream.

        :param name: The name of the stream.
        :return: Metadata about the stream.
        """
        try:
            response = self.kinesis_client.describe_stream(StreamName=name)
```

```
            self.name = name
            self.details = response["StreamDescription"]
            logger.info("Got stream %s.", name)
        except ClientError:
            logger.exception("Couldn't get %s.", name)
            raise
        else:
            return self.details
```

- Einzelheiten zur API finden Sie GetRecordsin AWS SDK for Python (Boto3) API Reference.

SAP ABAP

SDK für SAP ABAP

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
   TRY.
       oo_result = lo_kns->getrecords(             " oo_result is returned for
testing purposes. "
           iv_sharditerator = iv_shard_iterator ).
       DATA(lt_records) = oo_result->get_records( ).
       MESSAGE 'Record retrieved.' TYPE 'I'.
     CATCH /aws1/cx_knsexpirediteratorex.
       MESSAGE 'Iterator expired.' TYPE 'E'.
     CATCH /aws1/cx_knsinvalidargumentex.
       MESSAGE 'The specified argument was not valid.' TYPE 'E'.
     CATCH /aws1/cx_knskmsaccessdeniedex.
       MESSAGE 'You do not have permission to perform this AWS KMS action.' TYPE
'E'.
     CATCH /aws1/cx_knskmsdisabledex.
       MESSAGE 'KMS key used is disabled.' TYPE 'E'.
     CATCH /aws1/cx_knskmsinvalidstateex.
       MESSAGE 'KMS key used is in an invalid state. ' TYPE 'E'.
     CATCH /aws1/cx_knskmsnotfoundex.
       MESSAGE 'KMS key used is not found.' TYPE 'E'.
```

```
      CATCH /aws1/cx_knskmsoptinrequired.
        MESSAGE 'KMS key option is required.' TYPE 'E'.
      CATCH /aws1/cx_knskmsthrottlingex.
        MESSAGE 'The rate of requests to AWS KMS is exceeding the request
quotas.' TYPE 'E'.
      CATCH /aws1/cx_knsprovthruputexcdex.
        MESSAGE 'The request rate for the stream is too high, or the requested
data is too large for the available throughput.' TYPE 'E'.
      CATCH /aws1/cx_knsresourcenotfoundex.
        MESSAGE 'Resource being accessed is not found.' TYPE 'E'.
    ENDTRY.
```

- Einzelheiten zur API finden Sie GetRecordsin der API-Referenz zum AWS SDK für SAP ABAP.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Dienst mit einem SDK AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung von **`GetShardIterator`** mit einer CLI

Die folgenden Code-Beispiele zeigen, wie `GetShardIterator` verwendet wird.

Beispiele für Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Im folgenden Codebeispiel können Sie diese Aktion im Kontext sehen:

- Kennenlernen der Grundlagen

CLI

AWS CLI

So rufen Sie einen Shard-Iterator ab

Im folgenden Beispiel für `get-shard-iterator` wird der Shard-Iteratortyp `AT_SEQUENCE_NUMBER` verwendet und ein Shard-Iterator generiert, um mit dem Lesen von Datensätzen genau an der Position zu beginnen, die durch die angegebene Sequenznummer gekennzeichnet ist.

```
aws kinesis get-shard-iterator \
```

```
    --stream-name samplestream \
    --shard-id shardId-000000000001 \
    --shard-iterator-type LATEST
```

Ausgabe:

```
{
    "ShardIterator": "AAAAAAAAAAFEvJjIYI+3jw/4aqgH9FifJ+n48XWTh/
IFIsbILP6o5eDueD39NXNBfpZ10WL5K6ADXk8w+5H+Qhd9cFA9k268CPXCz/kebq1TGYI7Vy
+lUkA9BuN3xvATxMBGxRY3zYK05gqgvaIRn94O8SqeEqwhigwZxNWxID3Ej7YYYcxQi8Q/fIrCjGAy/
n2r5Z9G864YpWDfN9upNNQAR/iiOWKs"
}
```

Weitere Informationen finden Sie unter Developing Consumer Using the Kinesis Data Streams API with the AWS SDK for Java im Amazon Kinesis Data Streams Developer Guide.

- Einzelheiten zur API finden Sie GetShardIteratorin der AWS CLI Befehlsreferenz.

PowerShell

Tools für PowerShell V4

Beispiel 1: Gibt einen Shard-Iterator für den angegebenen Shard und die Startposition zurück. Einzelheiten zu den Shard-IDs und Sequenznummern können der Ausgabe des Get-KINStream Cmdlets entnommen werden, indem auf die Shards-Auflistung des zurückgegebenen Stream-Objekts verwiesen wird. Der zurückgegebene Iterator kann zusammen mit dem Get-KINRecord Cmdlet verwendet werden, um Datensätze im Shard abzurufen.

```
Get-KINShardIterator -StreamName "mystream" -ShardId "shardId-000000000000" -
ShardIteratorType AT_SEQUENCE_NUMBER -StartingSequenceNumber "495598645..."
```

Ausgabe:

```
AAAAAAAAAAGIc....9VnbiRNaP
```

- Einzelheiten zur API finden Sie unter AWS -Tools für PowerShell Cmdlet-Referenz (GetShardIteratorV4).

Tools für V5 PowerShell

Beispiel 1: Gibt einen Shard-Iterator für den angegebenen Shard und die Startposition zurück. Einzelheiten zu den Shard-IDs und Sequenznummern können der Ausgabe des Get-KINStream Cmdlets entnommen werden, indem auf die Shards-Auflistung des zurückgegebenen Stream-Objekts verwiesen wird. Der zurückgegebene Iterator kann zusammen mit dem Get-KINRecord Cmdlet verwendet werden, um Datensätze im Shard abzurufen.

```
Get-KINShardIterator -StreamName "mystream" -ShardId "shardId-000000000000" -
ShardIteratorType AT_SEQUENCE_NUMBER -StartingSequenceNumber "495598645..."
```

Ausgabe:

```
AAAAAAAAAAGIc....9VnbiRNaP
```

- Einzelheiten zur API finden Sie unter AWS -Tools für PowerShell Cmdlet-Referenz (GetShardIteratorV5).

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unter. Verwenden Sie diesen Dienst mit einem SDK AWS Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

`ListStreamConsumers`Mit einem AWS SDK verwenden

Das folgende Codebeispiel zeigt, wie es verwendet wird`ListStreamConsumers`.

.NET

SDK für .NET

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
using System;
using System.Collections.Generic;
```

```
    using System.Threading.Tasks;
    using Amazon.Kinesis;
    using Amazon.Kinesis.Model;

    /// <summary>
    /// List the consumers of an Amazon Kinesis stream.
    /// </summary>
    public class ListConsumers
    {
        public static async Task Main()
        {
            IAmazonKinesis client = new AmazonKinesisClient();

            string streamARN = "arn:aws:kinesis:us-east-2:000000000000:stream/
AmazonKinesisStream";
            int maxResults = 10;

            var consumers = await ListConsumersAsync(client, streamARN,
 maxResults);

            if (consumers.Count > 0)
            {
                consumers
                    .ForEach(c => Console.WriteLine($"Name: {c.ConsumerName} ARN:
 {c.ConsumerARN}"));
            }
            else
            {
                Console.WriteLine("No consumers found.");
            }
        }

        /// <summary>
        /// Retrieve a list of the consumers for a Kinesis stream.
        /// </summary>
        /// <param name="client">An initialized Kinesis client object.</param>
        /// <param name="streamARN">The ARN of the stream for which we want to
        /// retrieve a list of clients.</param>
        /// <param name="maxResults">The maximum number of results to return.</
param>
        /// <returns>A list of Consumer objects.</returns>
        public static async Task<List<Consumer>>
 ListConsumersAsync(IAmazonKinesis client, string streamARN, int maxResults)
        {
```

```
            var request = new ListStreamConsumersRequest
            {
                StreamARN = streamARN,
                MaxResults = maxResults,
            };

            var response = await client.ListStreamConsumersAsync(request);

            return response.Consumers;
        }
    }
```

- Einzelheiten zur API finden Sie ListStreamConsumersin der AWS SDK für .NET API-Referenz.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Dienst mit einem SDK AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`ListStreams`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `ListStreams` verwendet wird.

.NET

SDK für .NET

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.Kinesis;
using Amazon.Kinesis.Model;
```

```
    /// <summary>
    /// Retrieves and displays a list of existing Amazon Kinesis streams.
    /// </summary>
    public class ListStreams
    {
        public static async Task Main(string[] args)
        {
            IAmazonKinesis client = new AmazonKinesisClient();
            var response = await client.ListStreamsAsync(new
ListStreamsRequest());

            List<string> streamNames = response.StreamNames;

            if (streamNames.Count > 0)
            {
                streamNames
                    .ForEach(s => Console.WriteLine($"Stream name: {s}"));
            }
            else
            {
                Console.WriteLine("No streams were found.");
            }
        }
    }
```

- Einzelheiten zur API finden Sie ListStreamsin der AWS SDK für .NET API-Referenz.

CLI

AWS CLI

So listen Sie Datenströme auf

Im folgenden Beispiel für `list-streams` werden alle Parameter im aktuellen Konto und in der aktuellen Region aufgelistet.

```
aws kinesis list-streams
```

Ausgabe:

```
{
    "StreamNames": [
        "samplestream",
        "samplestream1"
    ]
}
```

Weitere Informationen finden Sie unter Auflisten eines Streams im Entwicklerhandbuch für Amazon Kinesis Data Streams.

- Einzelheiten zur API finden Sie ListStreamsin der AWS CLI Befehlsreferenz.

Rust

SDK für Rust

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
async fn show_streams(client: &Client) -> Result<(), Error> {
    let resp = client.list_streams().send().await?;

    println!("Stream names:");

    let streams = resp.stream_names;
    for stream in &streams {
        println!("  {}", stream);
    }

    println!("Found {} stream(s)", streams.len());

    Ok(())
}
```

- Einzelheiten zur API finden Sie ListStreamsin der API-Referenz zum AWS SDK für Rust.

SAP ABAP

SDK für SAP ABAP

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
   TRY.
       oo_result = lo_kns->liststreams(        " oo_result is returned for
testing purposes. "
           "Set Limit to specify that a maximum of streams should be returned."
           iv_limit = iv_limit ).
       DATA(lt_streams) = oo_result->get_streamnames( ).
       MESSAGE 'Streams listed.' TYPE 'I'.
     CATCH /aws1/cx_knslimitexceededex.
       MESSAGE 'The request processing has failed because of a limit exceed
exception.' TYPE 'E'.
   ENDTRY.
```

- Einzelheiten zur API finden Sie ListStreamsin der API-Referenz zum AWS SDK für SAP ABAP.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Dienst mit einem SDK AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`ListTagsForStream`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `ListTagsForStream` verwendet wird.

.NET

SDK für .NET

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    using System;
    using System.Collections.Generic;
    using System.Threading.Tasks;
    using Amazon.Kinesis;
    using Amazon.Kinesis.Model;

    /// <summary>
    /// Shows how to list the tags that have been attached to an Amazon Kinesis
    /// stream.
    /// </summary>
    public class ListTags
    {
        public static async Task Main()
        {
            IAmazonKinesis client = new AmazonKinesisClient();
            string streamName = "AmazonKinesisStream";

            await ListTagsAsync(client, streamName);
        }

        /// <summary>
        /// List the tags attached to a Kinesis stream.
        /// </summary>
        /// <param name="client">An initialized Kinesis client object.</param>
        /// <param name="streamName">The name of the Kinesis stream for which you
        /// wish to display tags.</param>
        public static async Task ListTagsAsync(IAmazonKinesis client, string
streamName)
        {
            var request = new ListTagsForStreamRequest
            {
                StreamName = streamName,
```

```
                Limit = 10,
            };

            var response = await client.ListTagsForStreamAsync(request);
            DisplayTags(response.Tags);

            while (response.HasMoreTags)
            {
                request.ExclusiveStartTagKey = response.Tags[response.Tags.Count
- 1].Key;
                response = await client.ListTagsForStreamAsync(request);
            }
        }

        /// <summary>
        /// Displays the items in a list of Kinesis tags.
        /// </summary>
        /// <param name="tags">A list of the Tag objects to be displayed.</param>
        public static void DisplayTags(List<Tag> tags)
        {
            tags
                .ForEach(t => Console.WriteLine($"Key: {t.Key} Value:
{t.Value}"));
        }
    }
```

- Einzelheiten zur API finden Sie ListTagsForStreamin der AWS SDK für .NET API-Referenz.

CLI

AWS CLI

So listen Sie Tags für einen Datenstrom auf

Im folgenden Beispiel für `list-tags-for-stream` werden alle Tags aufgelistet, die dem angegebenen Datenstrom zugeordnet sind.

```
aws kinesis list-tags-for-stream \
    --stream-name samplestream
```

Ausgabe:

```
{
    "Tags": [
        {
            "Key": "samplekey",
            "Value": "example"
        }
    ],
    "HasMoreTags": false
}
```

Weitere Informationen finden Sie unter Markieren Ihrer Streams im Entwicklerhandbuch für Amazon Kinesis Data Streams.

- Einzelheiten zur API finden Sie ListTagsForStreamin der AWS CLI Befehlsreferenz.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Dienst mit einem SDK AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`PutRecord`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `PutRecord` verwendet wird.

Beispiele für Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Im folgenden Codebeispiel können Sie diese Aktion im Kontext sehen:

- Kennenlernen der Grundlagen

CLI

AWS CLI

So schreiben Sie einen Datensatz in einen Datenstrom

Im folgenden Beispiel für `put-record` wird ein einzelner Datensatz mithilfe des angegebenen Partitionsschlüssels in den angegebenen Datenstrom geschrieben.

```
aws kinesis put-record \
    --stream-name samplestream \
```

```
    --data sampledatarecord \
    --partition-key samplepartitionkey
```

Ausgabe:

```
{
    "ShardId": "shardId-000000000009",
    "SequenceNumber": "49600902273357540915989931256901506243878407835297513618",
    "EncryptionType": "KMS"
}
```

Weitere Informationen finden Sie unter Entwickeln von Produzenten, die die Amazon Kinesis Data Streams-API mit dem AWS SDK for Java verwenden im Amazon Kinesis Data Streams Developer Guide.

- Einzelheiten zur API finden Sie PutRecordin der AWS CLI Befehlsreferenz.

Java

SDK für Java 2.x

> **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 software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.kinesis.KinesisClient;
import software.amazon.awssdk.services.kinesis.model.PutRecordRequest;
import software.amazon.awssdk.services.kinesis.model.KinesisException;
import software.amazon.awssdk.services.kinesis.model.DescribeStreamRequest;
import software.amazon.awssdk.services.kinesis.model.DescribeStreamResponse;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
```

```
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class StockTradesWriter {
    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <streamName>

                Where:
                    streamName - The Amazon Kinesis data stream to which records
 are written (for example, StockTradeStream)
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String streamName = args[0];
        Region region = Region.US_EAST_1;
        KinesisClient kinesisClient = KinesisClient.builder()
                .region(region)
                .build();

        // Ensure that the Kinesis Stream is valid.
        validateStream(kinesisClient, streamName);
        setStockData(kinesisClient, streamName);
        kinesisClient.close();
    }

    public static void setStockData(KinesisClient kinesisClient, String
 streamName) {
        try {
            // Repeatedly send stock trades with a 100 milliseconds wait in
 between.
            StockTradeGenerator stockTradeGenerator = new StockTradeGenerator();

            // Put in 50 Records for this example.
            int index = 50;
            for (int x = 0; x < index; x++) {
                StockTrade trade = stockTradeGenerator.getRandomTrade();
                sendStockTrade(trade, kinesisClient, streamName);
```

```
                Thread.sleep(100);
            }

        } catch (KinesisException | InterruptedException e) {
            System.err.println(e.getMessage());
            System.exit(1);
        }
        System.out.println("Done");
    }

    private static void sendStockTrade(StockTrade trade, KinesisClient
kinesisClient,
            String streamName) {
        byte[] bytes = trade.toJsonAsBytes();

        // The bytes could be null if there is an issue with the JSON
serialization by
        // the Jackson JSON library.
        if (bytes == null) {
            System.out.println("Could not get JSON bytes for stock trade");
            return;
        }

        System.out.println("Putting trade: " + trade);
        PutRecordRequest request = PutRecordRequest.builder()
                .partitionKey(trade.getTickerSymbol()) // We use the ticker
symbol as the partition key, explained in
                                                       // the Supplemental
Information section below.
                .streamName(streamName)
                .data(SdkBytes.fromByteArray(bytes))
                .build();

        try {
            kinesisClient.putRecord(request);
        } catch (KinesisException e) {
            System.err.println(e.getMessage());
        }
    }

    private static void validateStream(KinesisClient kinesisClient, String
streamName) {
        try {
```

```
            DescribeStreamRequest describeStreamRequest =
 DescribeStreamRequest.builder()
                    .streamName(streamName)
                    .build();

            DescribeStreamResponse describeStreamResponse =
 kinesisClient.describeStream(describeStreamRequest);

            if (!
describeStreamResponse.streamDescription().streamStatus().toString().equals("ACTIVE"))
 {
                System.err.println("Stream " + streamName + " is not active.
 Please wait a few moments and try again.");
                System.exit(1);
            }

        } catch (KinesisException e) {
            System.err.println("Error found while describing the stream " +
 streamName);
            System.err.println(e);
            System.exit(1);
        }
    }
}
```

- Einzelheiten zur API finden Sie PutRecordin der AWS SDK for Java 2.x API-Referenz.

PowerShell

Tools für PowerShell V4

Beispiel 1: Schreibt einen Datensatz, der die an den Parameter -Text übergebene Zeichenfolge enthält.

```
Write-KINRecord -Text "test data from string" -StreamName "mystream" -
PartitionKey "Key1"
```

Beispiel 2: Schreibt einen Datensatz, der die Daten aus der angegebenen Datei enthält. Die Datei wird als Folge von Bytes behandelt. Wenn sie Text enthält, sollte sie daher vor der Verwendung mit diesem Cmdlet mit der erforderlichen Kodierung geschrieben werden.

```
Write-KINRecord -FilePath "C:\TestData.txt" -StreamName "mystream" -PartitionKey
 "Key2"
```

- Einzelheiten zur API finden Sie unter PutRecord AWS -Tools für PowerShellCmdlet-Referenz (V4).

Tools für V5 PowerShell

Beispiel 1: Schreibt einen Datensatz, der die an den Parameter -Text übergebene Zeichenfolge enthält.

```
Write-KINRecord -Text "test data from string" -StreamName "mystream" -
PartitionKey "Key1"
```

Beispiel 2: Schreibt einen Datensatz, der die Daten aus der angegebenen Datei enthält. Die Datei wird als Folge von Bytes behandelt. Wenn sie Text enthält, sollte sie daher vor der Verwendung mit diesem Cmdlet mit der erforderlichen Kodierung geschrieben werden.

```
Write-KINRecord -FilePath "C:\TestData.txt" -StreamName "mystream" -PartitionKey
 "Key2"
```

- Einzelheiten zur API finden Sie unter PutRecord AWS -Tools für PowerShellCmdlet-Referenz (V5).

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
class KinesisStream:
    """Encapsulates a Kinesis stream."""

    def __init__(self, kinesis_client):
        """
        :param kinesis_client: A Boto3 Kinesis client.
```

```
        """
        self.kinesis_client = kinesis_client
        self.name = None
        self.details = None
        self.stream_exists_waiter = kinesis_client.get_waiter("stream_exists")


    def put_record(self, data, partition_key):
        """
        Puts data into the stream. The data is formatted as JSON before it is
passed
        to the stream.

        :param data: The data to put in the stream.
        :param partition_key: The partition key to use for the data.
        :return: Metadata about the record, including its shard ID and sequence
number.
        """
        try:
            response = self.kinesis_client.put_record(
                StreamName=self.name, Data=json.dumps(data),
PartitionKey=partition_key
            )
            logger.info("Put record in stream %s.", self.name)
        except ClientError:
            logger.exception("Couldn't put record in stream %s.", self.name)
            raise
        else:
            return response
```

- Einzelheiten zur API finden Sie PutRecordin AWS SDK for Python (Boto3) API Reference.

Rust

SDK für Rust

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
async fn add_record(client: &Client, stream: &str, key: &str, data: &str) ->
 Result<(), Error> {
    let blob = Blob::new(data);

    client
        .put_record()
        .data(blob)
        .partition_key(key)
        .stream_name(stream)
        .send()
        .await?;

    println!("Put data into stream.");

    Ok(())
}
```

- Einzelheiten zur API finden Sie PutRecordin der API-Referenz zum AWS SDK für Rust.

SAP ABAP

SDK für SAP ABAP

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    TRY.
        oo_result = lo_kns->putrecord(            " oo_result is returned for
 testing purposes. "
            iv_streamname = iv_stream_name
            iv_data       = iv_data
            iv_partitionkey = iv_partition_key ).
        MESSAGE 'Record created.' TYPE 'I'.
      CATCH /aws1/cx_knsinvalidargumentex.
        MESSAGE 'The specified argument was not valid.' TYPE 'E'.
      CATCH /aws1/cx_knskmsaccessdeniedex.
```

```
        MESSAGE 'You do not have permission to perform this AWS KMS action.' TYPE
'E'.
      CATCH /aws1/cx_knskmsdisabledex.
        MESSAGE 'KMS key used is disabled.' TYPE 'E'.
      CATCH /aws1/cx_knskmsinvalidstateex.
        MESSAGE 'KMS key used is in an invalid state. ' TYPE 'E'.
      CATCH /aws1/cx_knskmsnotfoundex.
        MESSAGE 'KMS key used is not found.' TYPE 'E'.
      CATCH /aws1/cx_knskmsoptinrequired.
        MESSAGE 'KMS key option is required.' TYPE 'E'.
      CATCH /aws1/cx_knskmsthrottlingex.
        MESSAGE 'The rate of requests to AWS KMS is exceeding the request
quotas.' TYPE 'E'.
      CATCH /aws1/cx_knsprovthruputexcdex.
        MESSAGE 'The request rate for the stream is too high, or the requested
data is too large for the available throughput.' TYPE 'E'.
      CATCH /aws1/cx_knsresourcenotfoundex.
        MESSAGE 'Resource being accessed is not found.' TYPE 'E'.
    ENDTRY.
```

- Einzelheiten zur API finden Sie PutRecordin der API-Referenz zum AWS SDK für SAP ABAP.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Dienst mit einem SDK AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`PutRecords`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `PutRecords` verwendet wird.

CLI

AWS CLI

So schreiben Sie mehrere Datensätze in einen Datenstrom

Im folgenden Beispiel für `put-records` werden ein Datensatz mit dem angegebenen Partitionsschlüssel und ein anderer Datensatz mit einem anderen Partitionsschlüssel in einen einzelnen Aufruf geschrieben.

```
aws kinesis put-records \
    --stream-name samplestream \
    --
records Data=blob1,PartitionKey=partitionkey1 Data=blob2,PartitionKey=partitionkey2
```

Ausgabe:

```
{
    "FailedRecordCount": 0,
    "Records": [
        {
            "SequenceNumber":
 "49600883331171471519674795588238531498465399900093808706",
            "ShardId": "shardId-000000000004"
        },
        {
            "SequenceNumber":
 "49600902273357540915989931256902715169698037101720764562",
            "ShardId": "shardId-000000000009"
        }
    ],
    "EncryptionType": "KMS"
}
```

Weitere Informationen finden Sie unter Entwickeln von Produzenten, die die Amazon Kinesis Data Streams-API mit dem AWS SDK for Java verwenden im Amazon Kinesis Data Streams Developer Guide.

- Einzelheiten zur API finden Sie PutRecordsin der AWS CLI Befehlsreferenz.

JavaScript

SDK für JavaScript (v3)

> **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
             * 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 PutRecordsin der AWS SDK für JavaScript API-Referenz.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Dienst mit einem SDK AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Verwendung **`RegisterStreamConsumer`** mit einem AWS SDK oder CLI

Die folgenden Code-Beispiele zeigen, wie `RegisterStreamConsumer` verwendet wird.

.NET

SDK für .NET

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
using System;
using System.Threading.Tasks;
using Amazon.Kinesis;
using Amazon.Kinesis.Model;
```

```
    /// <summary>
    /// This example shows how to register a consumer to an Amazon Kinesis
    /// stream.
    /// </summary>
    public class RegisterConsumer
    {
        public static async Task Main()
        {
            IAmazonKinesis client = new AmazonKinesisClient();
            string consumerName = "NEW_CONSUMER_NAME";
            string streamARN = "arn:aws:kinesis:us-east-2:000000000000:stream/
AmazonKinesisStream";

            var consumer = await RegisterConsumerAsync(client, consumerName,
 streamARN);

            if (consumer is not null)
            {
                Console.WriteLine($"{consumer.ConsumerName}");
            }
        }

        /// <summary>
        /// Registers the consumer to a Kinesis stream.
        /// </summary>
        /// <param name="client">The initialized Kinesis client object.</param>
        /// <param name="consumerName">A string representing the consumer.</
param>
        /// <param name="streamARN">The ARN of the stream.</param>
        /// <returns>A Consumer object that contains information about the
 consumer.</returns>
        public static async Task<Consumer> RegisterConsumerAsync(IAmazonKinesis
 client, string consumerName, string streamARN)
        {
            var request = new RegisterStreamConsumerRequest
            {
                ConsumerName = consumerName,
                StreamARN = streamARN,
            };

            var response = await client.RegisterStreamConsumerAsync(request);
            return response.Consumer;
        }
    }
```

- Einzelheiten zur API finden Sie RegisterStreamConsumerin der AWS SDK für .NET API-Referenz.

CLI

AWS CLI

So registrieren Sie einen Datenstromkonsumenten

Im folgenden Beispiel für `register-stream-consumer` wird ein Konsument namens `KinesisConsumerApplication` beim angegebenen Datenstrom registriert.

```
aws kinesis register-stream-consumer \
    --stream-arn arn:aws:kinesis:us-west-2:012345678912:stream/samplestream \
    --consumer-name KinesisConsumerApplication
```

Ausgabe:

```
{
    "Consumer": {
        "ConsumerName": "KinesisConsumerApplication",
        "ConsumerARN": "arn:aws:kinesis:us-west-2: 123456789012:stream/
samplestream/consumer/KinesisConsumerApplication:1572383852",
        "ConsumerStatus": "CREATING",
        "ConsumerCreationTimestamp": 1572383852.0
    }
}
```

Weitere Informationen finden Sie unter Entwickeln von Konsumenten mit Enhanced Fan-Out über die API von Kinesis Data Streams im Entwicklerhandbuch für Amazon Kinesis Data Streams.

- Einzelheiten zur API finden Sie RegisterStreamConsumerin der AWS CLI Befehlsreferenz.

SAP ABAP

SDK für SAP ABAP

> **Note**
>
> Es gibt noch mehr dazu. GitHub Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
   TRY.
       oo_result = lo_kns->registerstreamconsumer(       " oo_result is returned
for testing purposes. "
           iv_streamarn = iv_stream_arn
           iv_consumername = iv_consumer_name ).
       MESSAGE 'Stream consumer registered.' TYPE 'I'.
     CATCH /aws1/cx_knsinvalidargumentex.
       MESSAGE 'The specified argument was not valid.' TYPE 'E'.
     CATCH /aws1/cx_sgmresourcelimitexcd.
       MESSAGE 'You have reached the limit on the number of resources.' TYPE
'E'.
     CATCH /aws1/cx_sgmresourceinuse.
       MESSAGE 'Resource being accessed is in use.' TYPE 'E'.
     CATCH /aws1/cx_sgmresourcenotfound.
       MESSAGE 'Resource being accessed is not found.' TYPE 'E'.
   ENDTRY.
```

- Einzelheiten zur API finden Sie RegisterStreamConsumerin der API-Referenz zum AWS SDK für SAP ABAP.

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Dienst mit einem SDK AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Serverless-Beispiele für Kinesis

Die folgenden Codebeispiele zeigen, wie Kinesis mit AWS SDKs verwendet wird.

Beispiele

- Aufrufen einer Lambda-Funktion über einen Kinesis-Auslöser
- Melden von Batch-Elementfehlern für Lambda-Funktionen mit einem Kinesis-Auslöser

Aufrufen einer Lambda-Funktion über einen Kinesis-Auslöser

Die folgenden Codebeispiele veranschaulichen, 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.

.NET

SDK für .NET

> **Note**
>
> Es gibt noch mehr dazu GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit Serverless-Beispielen.

Nutzen eines Kinesis-Ereignisses mit Lambda unter Verwendung von .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text;
using Amazon.Lambda.Core;
using Amazon.Lambda.KinesisEvents;
using AWS.Lambda.Powertools.Logging;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace KinesisIntegrationSampleCode;

public class Function
{
    // Powertools Logger requires an environment variables against your function
    // POWERTOOLS_SERVICE_NAME
    [Logging(LogEvent = true)]
    public async Task FunctionHandler(KinesisEvent evnt, ILambdaContext context)
```

```
    {
        if (evnt.Records.Count == 0)
        {
            Logger.LogInformation("Empty Kinesis Event received");
            return;
        }

        foreach (var record in evnt.Records)
        {
            try
            {
                Logger.LogInformation($"Processed Event with EventId:
 {record.EventId}");
                string data = await GetRecordDataAsync(record.Kinesis, context);
                Logger.LogInformation($"Data: {data}");
                // TODO: Do interesting work based on the new data
            }
            catch (Exception ex)
            {
                Logger.LogError($"An error occurred {ex.Message}");
                throw;
            }
        }
        Logger.LogInformation($"Successfully processed {evnt.Records.Count}
 records.");
    }

    private async Task<string> GetRecordDataAsync(KinesisEvent.Record record,
 ILambdaContext context)
    {
        byte[] bytes = record.Data.ToArray();
        string data = Encoding.UTF8.GetString(bytes);
        await Task.CompletedTask; //Placeholder for actual async work
        return data;
    }
}
```

Go

SDK für Go V2

> **Note**
>
> Es gibt noch mehr GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit Serverless-Beispielen.

Nutzen eines Kinesis-Ereignisses mit Lambda unter Verwendung von Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "log"

 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, kinesisEvent events.KinesisEvent) error {
 if len(kinesisEvent.Records) == 0 {
  log.Printf("empty Kinesis event received")
  return nil
 }

 for _, record := range kinesisEvent.Records {
  log.Printf("processed Kinesis event with EventId: %v", record.EventID)
  recordDataBytes := record.Kinesis.Data
  recordDataText := string(recordDataBytes)
  log.Printf("record data: %v", recordDataText)
  // TODO: Do interesting work based on the new data
 }
 log.Printf("successfully processed %v records", len(kinesisEvent.Records))
 return nil
}

func main() {
 lambda.Start(handler)
```

```
}
```

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit Serverless-Beispielen.

Nutzen eines Kinesis-Ereignisses mit Lambda unter Verwendung von Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package example;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.LambdaLogger;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.KinesisEvent;

public class Handler implements RequestHandler<KinesisEvent, Void> {
    @Override
    public Void handleRequest(final KinesisEvent event, final Context context) {
        LambdaLogger logger = context.getLogger();
        if (event.getRecords().isEmpty()) {
            logger.log("Empty Kinesis Event received");
            return null;
        }
        for (KinesisEvent.KinesisEventRecord record : event.getRecords()) {
            try {
                logger.log("Processed Event with EventId: "+record.getEventID());
                String data = new String(record.getKinesis().getData().array());
                logger.log("Data:"+ data);
                // TODO: Do interesting work based on the new data
            }
            catch (Exception ex) {
                logger.log("An error occurred:"+ex.getMessage());
                throw ex;
```

```
            }
        }
        logger.log("Successfully processed:"+event.getRecords().size()+"
 records");
        return null;
    }

}
```

JavaScript

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;
}
```

PHP

SDK für PHP

> **Note**
>
> Es gibt noch mehr dazu. GitHub Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit Serverless-Beispielen.

Nutzen eines Kinesis-Ereignisses mit Lambda unter Verwendung von PHP.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\Kinesis\KinesisEvent;
use Bref\Event\Kinesis\KinesisHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler extends KinesisHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
```

```
     */
    public function handleKinesis(KinesisEvent $event, Context $context): void
    {
        $this->logger->info("Processing records");
        $records = $event->getRecords();
        foreach ($records as $record) {
            $data = $record->getData();
            $this->logger->info(json_encode($data));
            // TODO: Do interesting work based on the new data

            // Any exception thrown will be logged and the invocation will be
 marked as failed
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords records");
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit Serverless-Beispielen.

Nutzen eines Kinesis-Ereignisses mit Lambda unter Verwendung von Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
import base64
def lambda_handler(event, context):

    for record in event['Records']:
        try:
            print(f"Processed Kinesis Event - EventID: {record['eventID']}")
```

```
            record_data = base64.b64decode(record['kinesis']
['data']).decode('utf-8')
            print(f"Record Data: {record_data}")
            # TODO: Do interesting work based on the new data
        except Exception as e:
            print(f"An error occurred {e}")
            raise e
    print(f"Successfully processed {len(event['Records'])} records.")
```

Ruby

SDK für Ruby

> **Note**
>
> Es gibt noch mehr GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit Serverless-Beispielen.

Nutzen eines Kinesis-Ereignisses mit Lambda unter Verwendung von Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
require 'aws-sdk'

def lambda_handler(event:, context:)
  event['Records'].each do |record|
    begin
      puts "Processed Kinesis Event - EventID: #{record['eventID']}"
      record_data = get_record_data_async(record['kinesis'])
      puts "Record Data: #{record_data}"
      # TODO: Do interesting work based on the new data
    rescue => err
      $stderr.puts "An error occurred #{err}"
      raise err
    end
  end
  puts "Successfully processed #{event['Records'].length} records."
end

def get_record_data_async(payload)
```

```
  data = Base64.decode64(payload['data']).force_encoding('UTF-8')
  # Placeholder for actual async work
  # You can use Ruby's asynchronous programming tools like async/await or fibers
 here.
  return data
end
```

Rust

SDK für Rust

> **Note**
>
> Es gibt noch mehr dazu GitHub. Das vollständige Beispiel sowie eine Anleitung zum Einrichten und Ausführen finden Sie im Repository mit Serverless-Beispielen.

Nutzen eines Kinesis-Ereignisses mit Lambda unter Verwendung von Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::event::kinesis::KinesisEvent;
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

async fn function_handler(event: LambdaEvent<KinesisEvent>) -> Result<(), Error>
 {
    if event.payload.records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(());
    }

    event.payload.records.iter().for_each(|record| {
        tracing::info!("EventId:
 {}",record.event_id.as_deref().unwrap_or_default());

        let record_data = std::str::from_utf8(&record.kinesis.data);

        match record_data {
            Ok(data) => {
                // log the record data
                tracing::info!("Data: {}", data);
            }
```

```
            Err(e) => {
                tracing::error!("Error: {}", e);
            }
        }
    });

    tracing::info!(
        "Successfully processed {} records",
        event.payload.records.len()
    );

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
        // disabling time is handy because CloudWatch will add the ingestion
 time.
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}
```

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Dienst mit einem SDK AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Melden von Batch-Elementfehlern für Lambda-Funktionen mit einem Kinesis-Auslöser

Die folgenden Codebeispiele zeigen, 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.

.NET

SDK für .NET

> **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 .NET.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
using System.Text;
using System.Text.Json.Serialization;
using Amazon.Lambda.Core;
using Amazon.Lambda.KinesisEvents;
using AWS.Lambda.Powertools.Logging;

// Assembly attribute to enable the Lambda function's JSON input to be converted
 into a .NET class.
[assembly:
 LambdaSerializer(typeof(Amazon.Lambda.Serialization.SystemTextJson.DefaultLambdaJsonSeri

namespace KinesisIntegration;

public class Function
{
    // Powertools Logger requires an environment variables against your function
    // POWERTOOLS_SERVICE_NAME
    [Logging(LogEvent = true)]
    public async Task<StreamsEventResponse> FunctionHandler(KinesisEvent evnt,
 ILambdaContext context)
    {
        if (evnt.Records.Count == 0)
        {
            Logger.LogInformation("Empty Kinesis Event received");
            return new StreamsEventResponse();
        }

        foreach (var record in evnt.Records)
        {
```

```
            try
            {
                Logger.LogInformation($"Processed Event with EventId:
 {record.EventId}");
                string data = await GetRecordDataAsync(record.Kinesis, context);
                Logger.LogInformation($"Data: {data}");
                // TODO: Do interesting work based on the new data
            }
            catch (Exception ex)
            {
                Logger.LogError($"An error occurred {ex.Message}");
                /* Since we are working with streams, we can return the failed
 item immediately.
                   Lambda will immediately begin to retry processing from this
 failed item onwards. */
                return new StreamsEventResponse
                {
                    BatchItemFailures = new
 List<StreamsEventResponse.BatchItemFailure>
                    {
                        new StreamsEventResponse.BatchItemFailure
 { ItemIdentifier = record.Kinesis.SequenceNumber }
                    }
                };
            }
        }
        Logger.LogInformation($"Successfully processed {evnt.Records.Count}
 records.");
        return new StreamsEventResponse();
    }

    private async Task<string> GetRecordDataAsync(KinesisEvent.Record record,
 ILambdaContext context)
    {
        byte[] bytes = record.Data.ToArray();
        string data = Encoding.UTF8.GetString(bytes);
        await Task.CompletedTask; //Placeholder for actual async work
        return data;
    }
}

public class StreamsEventResponse
{
    [JsonPropertyName("batchItemFailures")]
```

```
    public IList<BatchItemFailure> BatchItemFailures { get; set; }
    public class BatchItemFailure
    {
        [JsonPropertyName("itemIdentifier")]
        public string ItemIdentifier { get; set; }
    }
}
```

Go

SDK für Go V2

> **Note**
>
> Es gibt noch mehr 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 Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "fmt"
 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, kinesisEvent events.KinesisEvent)
 (map[string]interface{}, error) {
 batchItemFailures := []map[string]interface{}{}

 for _, record := range kinesisEvent.Records {
  curRecordSequenceNumber := ""

  // Process your record
  if /* Your record processing condition here */ {
   curRecordSequenceNumber = record.Kinesis.SequenceNumber
  }
```

```
  // Add a condition to check if the record processing failed
  if curRecordSequenceNumber != "" {
   batchItemFailures = append(batchItemFailures, map[string]interface{}
{"itemIdentifier": curRecordSequenceNumber})
  }
 }

 kinesisBatchResponse := map[string]interface{}{
  "batchItemFailures": batchItemFailures,
 }
 return kinesisBatchResponse, nil
}

func main() {
 lambda.Start(handler)
}
```

Java

SDK für Java 2.x

> **Note**
>
> Es gibt noch mehr 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 Java.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;
import com.amazonaws.services.lambda.runtime.events.KinesisEvent;
import com.amazonaws.services.lambda.runtime.events.StreamsEventResponse;

import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;
```

```
public class ProcessKinesisRecords implements RequestHandler<KinesisEvent,
 StreamsEventResponse> {

    @Override
    public StreamsEventResponse handleRequest(KinesisEvent input, Context
 context) {

        List<StreamsEventResponse.BatchItemFailure> batchItemFailures = new
 ArrayList<>();
        String curRecordSequenceNumber = "";

        for (KinesisEvent.KinesisEventRecord kinesisEventRecord :
 input.getRecords()) {
            try {
                //Process your record
                KinesisEvent.Record kinesisRecord =
 kinesisEventRecord.getKinesis();
                curRecordSequenceNumber = kinesisRecord.getSequenceNumber();

            } catch (Exception e) {
                /* 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. */
                batchItemFailures.add(new
 StreamsEventResponse.BatchItemFailure(curRecordSequenceNumber));
                return new StreamsEventResponse(batchItemFailures);
            }
        }

       return new StreamsEventResponse(batchItemFailures);
    }
}
```

JavaScript

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;
}
```

PHP

SDK für PHP

> **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 PHP.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
<?php

# using bref/bref and bref/logger for simplicity

use Bref\Context\Context;
use Bref\Event\Kinesis\KinesisEvent;
use Bref\Event\Handler as StdHandler;
use Bref\Logger\StderrLogger;

require __DIR__ . '/vendor/autoload.php';

class Handler implements StdHandler
{
    private StderrLogger $logger;
    public function __construct(StderrLogger $logger)
```

```
    {
        $this->logger = $logger;
    }

    /**
     * @throws JsonException
     * @throws \Bref\Event\InvalidLambdaEvent
     */
    public function handle(mixed $event, Context $context): array
    {
        $kinesisEvent = new KinesisEvent($event);
        $this->logger->info("Processing records");
        $records = $kinesisEvent->getRecords();

        $failedRecords = [];
        foreach ($records as $record) {
            try {
                $data = $record->getData();
                $this->logger->info(json_encode($data));
                // TODO: Do interesting work based on the new data
            } catch (Exception $e) {
                $this->logger->error($e->getMessage());
                // failed processing the record
                $failedRecords[] = $record->getSequenceNumber();
            }
        }
        $totalRecords = count($records);
        $this->logger->info("Successfully processed $totalRecords records");

        // change format for the response
        $failures = array_map(
            fn(string $sequenceNumber) => ['itemIdentifier' => $sequenceNumber],
            $failedRecords
        );

        return [
            'batchItemFailures' => $failures
        ];
    }
}

$logger = new StderrLogger();
return new Handler($logger);
```

Python

SDK für Python (Boto3)

> **Note**
>
> Es gibt noch mehr 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 Python.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
def handler(event, context):
    records = event.get("Records")
    curRecordSequenceNumber = ""

    for record in records:
        try:
            # Process your record
            curRecordSequenceNumber = record["kinesis"]["sequenceNumber"]
        except Exception as e:
            # Return failed record's sequence number
            return {"batchItemFailures":[{"itemIdentifier":
 curRecordSequenceNumber}]}

    return {"batchItemFailures":[]}
```

Ruby

SDK für Ruby

> **Note**
>
> Es gibt noch mehr 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 Ruby.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
require 'aws-sdk'

def lambda_handler(event:, context:)
  batch_item_failures = []

  event['Records'].each do |record|
    begin
      puts "Processed Kinesis Event - EventID: #{record['eventID']}"
      record_data = get_record_data_async(record['kinesis'])
      puts "Record Data: #{record_data}"
      # TODO: Do interesting work based on the new data
    rescue StandardError => err
      puts "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'] }] }
    end
  end

  puts "Successfully processed #{event['Records'].length} records."
  { batchItemFailures: batch_item_failures }
end

def get_record_data_async(payload)
  data = Base64.decode64(payload['data']).force_encoding('utf-8')
  # Placeholder for actual async work
```

```
  sleep(1)
  data
end
```

Rust

SDK für Rust

> **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 Rust.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
use aws_lambda_events::{
    event::kinesis::KinesisEvent,
    kinesis::KinesisEventRecord,
    streams::{KinesisBatchItemFailure, KinesisEventResponse},
};
use lambda_runtime::{run, service_fn, Error, LambdaEvent};

async fn function_handler(event: LambdaEvent<KinesisEvent>) ->
 Result<KinesisEventResponse, Error> {
    let mut response = KinesisEventResponse {
        batch_item_failures: vec![],
    };

    if event.payload.records.is_empty() {
        tracing::info!("No records found. Exiting.");
        return Ok(response);
    }

    for record in &event.payload.records {
        tracing::info!(
            "EventId: {}",
            record.event_id.as_deref().unwrap_or_default()
        );
```

```
        let record_processing_result = process_record(record);

        if record_processing_result.is_err() {
            response.batch_item_failures.push(KinesisBatchItemFailure {
                item_identifier: record.kinesis.sequence_number.clone(),
            });
            /* 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 Ok(response);
        }
    }

    tracing::info!(
        "Successfully processed {} records",
        event.payload.records.len()
    );

    Ok(response)
}

fn process_record(record: &KinesisEventRecord) -> Result<(), Error> {
    let record_data = std::str::from_utf8(record.kinesis.data.as_slice());

    if let Some(err) = record_data.err() {
        tracing::error!("Error: {}", err);
        return Err(Error::from(err));
    }

    let record_data = record_data.unwrap_or_default();

    // do something interesting with the data
    tracing::info!("Data: {}", record_data);

    Ok(())
}

#[tokio::main]
async fn main() -> Result<(), Error> {
    tracing_subscriber::fmt()
        .with_max_level(tracing::Level::INFO)
        // disable printing the name of the module in every log line.
        .with_target(false)
```

```
        // disabling time is handy because CloudWatch will add the ingestion
 time.
        .without_time()
        .init();

    run(service_fn(function_handler)).await
}
```

Eine vollständige Liste der AWS SDK-Entwicklerhandbücher und Codebeispiele finden Sie unterVerwenden Sie diesen Dienst mit einem SDK AWS. Dieses Thema enthält auch Informationen zu den ersten Schritten und Details zu früheren SDK-Versionen.

Dokumentverlauf

Die folgende Tabelle enthält wichtige Änderungen an der Dokumentation zu Amazon Kinesis Data Streams.

Änderungen	Beschreibung	Änderungsdatum
Unterstützung für Resilienztests mit hinzugefügt AWS Fault Injection Service.	Führen Sie Resilienztests durch mit AWS Fault Injection Service hinzugefügt.	15. Oktober 2025
Unterstützung für große Datensatz größen von bis zu 10 MiB hinzugefü gt.	Umgang mit großen Datensätzen hinzugefügt.	27. Oktober 2025
End-of-support Daten und Richtlini en zum Versionsl ebenszyklus für KPL und KCL.	Es wurden Informationen zu end-of-support Daten und Richtlinien für den Versionslebenszyklus für Amazon Kinesis Producer Library (KPL) und Amazon Kinesis Client Library (KCL) hinzugefügt. Weitere Informationen erhalten Sie unter Richtlinie zum Lebenszyklus der KPL-Version und Lebenszyklus-Richtlinie für KCL-Versionen.	13. März 2025
Unterstützung für die gemeinsam e Nutzung von Datenströmen zwischen Konten hinzugefügt.	Teilen Sie Ihren Datenstream mit einem anderen Konto hinzugefügt.	22. November 2023
Unterstützung für die Kapazitätsmodi „On-Demand“ und	Wählen Sie den richtigen Modus zum Streamen hinzugefügt.	29. November 2021

Änderungen	Beschreibung	Änderungsdatum
„Bereitgestellter Datenstrom“ wurde hinzugefügt.		
Neue Inhalte für die serverseitige Verschlüsselung.	Datenschutz in Amazon Kinesis Data Streams hinzugefügt.	7. Juli 2017
Neue Inhalte für verbesser te CloudWatch Metriken.	Aktualisiert Kinesis Data Streams überwachen.	19. April 2016
Neue Inhalte zur Erweiterung von Kinesis-Agenten.	Aktualisiert Schreiben Sie mit Kinesis Agent in Amazon Kinesis Data Streams.	11. April 2016
Neue Inhalte für die Nutzung von Kinesis-Agenten.	Schreiben Sie mit Kinesis Agent in Amazon Kinesis Data Streams hinzugefügt.	2. Oktober 2015
Aktualisierter KPL-Inhalt für Version 0.10.0.	Entwickeln Sie Produzenten mithilfe der Amazon Kinesis Producer Library (KPL) hinzugefügt.	15. Juli 2015
Aktualisierter Inhalt zum Thema KCL-Metriken für konfigurierbare Metriken.	Überwachen Sie die Kinesis-Clientbibliothek mit Amazon CloudWatch hinzugefügt.	9. Juli 2015
Inhalte neu organisiert.	Wichtige Neuorganisation von Inhalten für einen bessere Strukturansicht und eine logischere Gruppierung.	01. Juli 2015

Änderungen	Beschreibung	Änderungsdatum
Neues Thema für das KPL-Entwicklerhandbuch.	Entwickeln Sie Produzenten mithilfe der Amazon Kinesis Producer Library (KPL) hinzugefügt.	02. Juni 2015
Neues Thema zu KCL-Metriken.	Überwachen Sie die Kinesis-Clientbibliothek mit Amazon CloudWatch hinzugefügt.	19. Mai 2015
KCL-Support in .NET	Entwickeln Sie einen Kinesis Client Library-Consumer in .NET hinzugefügt.	1. Mai 2015
KCL-Support in Node.js	Entwickeln Sie einen Kinesis Client Library-Consumer in Node.js hinzugefügt.	26. März 2015
KCL-Support in Ruby	Links zur KCL Ruby-Bibliothek hinzugefügt.	12. Januar 2015
Neue API PutRecords	Informationen über die neue PutRecords API wurden hinzugefügtthe section called “Fügen Sie mehrere Datensätze hinzu mit PutRecords”.	15. Dezember 2014
Support für Markierungen	Kennzeichnen Sie Ihre Amazon Kinesis Data Streams Streams-Ressourcen hinzugefügt.	11. September 2014
Neue CloudWatch Metrik	`GetRecords.IteratorAgeMilliseconds` - Metrik zu Dimensionen und Metriken von Amazon Kinesis Data Streams hinzugefügt.	3. September 2014
Neues Kapitel zur Überwachung	Kinesis Data Streams überwachen und Überwachen Sie den Amazon Kinesis Data Streams Streams-Service mit Amazon CloudWatch hinzugefügt.	30. Juli 2014
Standard-Shard-Limit	Kontingente und -Einschränkungen aktualisiert: Das Standard-Shard-Limit wurde von 5 auf 10 erhöht.	25. Februar 2014
Standard-Shard-Limit	Kontingente und -Einschränkungen aktualisiert: Das Standard-Shard-Limit wurde von 2 auf 5 erhöht.	28. Januar 2014

Änderungen	Beschreibung	Änderungsdatum
Aktualisierungen der API-Version	Aktualisierungen von Version 2013-12-02 der API für Kinesis Data Streams.	12. Dezember 2013
Erstversion	Erstveröffentlichung des Entwicklerhandbuchs für Amazon Kinesis.	14. November 2013

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.