Guia do desenvolvedor

AWS SDK para .NET (V4)

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

AWS SDK para .NET (V4): Guia do desenvolvedor

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

As marcas comerciais e imagens de marcas da Amazon não podem ser usadas no contexto de nenhum produto ou serviço que não seja da Amazon, nem de qualquer maneira que possa gerar confusão entre os clientes ou que deprecie ou desprestigie a Amazon. Todas as outras marcas comerciais que não pertencem à Amazon pertencem a seus respectivos proprietários, que podem ou não ser afiliados, patrocinados pela Amazon ou ter conexão com ela.

Table of Contents

..... x
O que é o AWS SDK para .NET 1
Sobre essa versão 1
Manutenção e suporte para as versões principais do SDK 2
Casos de uso comuns 2
Tópicos adicionais nesta seção 2
AWS Ferramentas relacionadas 3
Ferramentas para Windows PowerShell e Ferramentas para PowerShell Core 3
Toolkit for VS Code 3
Toolkit para Visual Studio 3
Kit de ferramentas para Azure DevOps 4
Plataformas compatíveis 4
.NET Core 4
.NET Standard 2.0 4
.NET Framework 4.5 5
.NET Framework 3.5 5
Biblioteca de classes portátil e Xamarin 5
Suporte ao Unity 5
SDKs Referência de ferramentas e ferramentas 5
Histórico de revisões 6
O que há de novo 6
Recursos adicionais 8
Conceitos básicos 10
Instalando e configurando seu conjunto de ferramentas 10
Desenvolvimento entre plataformas 10
Windows com Visual Studio e .NET Core 11
Próxima etapa 11
Autenticando com AWS 11
Faça login usando as credenciais do console 12
Habilitar e configurar o Centro de Identidade do IAM 13
Configure o SDK para usar o Centro de Identidade do IAM. 13
Iniciar uma sessão do portal de AWS acesso 14
Mais informações 15
Criando um aplicativo simples 16

Aplicativo simples para várias plataformas ... 16
Aplicativo simples baseado no Windows ... 22
Próximas etapas ... 28
Configuração ... 29
Iniciar um novo projeto ... 29
Instale AWSSDK pacotes com NuGet ... 31
Usando NuGet a partir do prompt de comando ou terminal ... 32
Usando NuGet do Visual Studio Solution Explorer ... 32
Usando NuGet a partir do Package Manager Console ... 33
Instale AWSSDK montagens sem NuGet ... 34
AWS Região ... 35
Crie um cliente de serviço com uma região específica ... 36
Especifique uma região para todos os clientes de serviço ... 37
Resolução da região ... 38
Informações especiais sobre a região da China (Pequim) ... 38
Informações especiais sobre novos AWS serviços ... 38
Resolução de perfil e credenciais ... 39
Resolução do perfil ... 40
Usar credenciais da conta de usuário federado ... 40
Especificar funções ou credenciais temporárias ... 41
Usar credenciais de proxy ... 41
Novas tentativas e tempos limite ... 42
Novas tentativas ... 42
Tempos limite ... 44
Observabilidade ... 47
Recursos adicionais do ... 47
Configurar um `TelemetryProvider` ... 47
Métricas ... 49
Provedores de telemetria ... 51
Usuários e perfis ... 52
Usuários e conjuntos de permissões ... 52
Perfis de serviço ... 53
Configurações avançadas ... 54
AWSSDK.Extensões. NETCore.Configuração e IConfiguration ... 54
Configuração de outros parâmetros do aplicativo ... 59
Referência de arquivos de configuração para AWS SDK para .NET ... 67

Uso de credenciais herdadas 79
Avisos e orientações importantes para credenciais 80
Usando o arquivo de AWS credenciais compartilhado 81
Usando a SDK Store (somente Windows) 84
Uso da SDK 89
Programação assíncrona 89
Paginação 91
Onde posso encontrar paginadores? 91
O que os paginadores me oferecem? 91
Paginação síncrona em comparação à assíncrona 92
Exemplo 92
Considerações adicionais sobre paginadores 96
Support para HTTP 2 96
Considerações adicionais 99
Ferramentas adicionais 99
AWS Ferramenta de implantação 99
AWS Estrutura de processamento de mensagens para.NET 100
Integrações com o.NET Aspire 100
Autenticação avançada 101
Autenticação única 101
Pré-requisitos 102
Como configurar um perfil de SSO 102
Como gerar e usar de tokens de SSO 104
Recursos adicionais 109
Tutoriais 109
Tutorial: somente aplicativo .NET 109
Tutorial: AWS CLI e aplicativo.NET 119
Implemente em AWS 128
Implantar a partir da CLI do .NET 128
Implante a partir dos kits de ferramentas IDE 128
Casos de uso 129
Aplicativos ASP.NET Core: 129
Aplicativos do .NET Console 130
Aplicativos Blazor WebAssembly 131
AWS Lambda projetos 132
Pré-requisitos 132

Comandos do Lambda disponíveis 132
Etapas para implantar 133
AWS Serviços de chamadas 135
Exemplos de código guiado 135
CloudFormation 136
Amazon Cognito 140
DynamoDB 149
Amazon EC2 179
IAM 241
Amazon S3 261
Amazon SNS 271
Amazon SQS 275
AWS Lambda 308
APIs 308
Pré-requisitos 308
Mais informações 309
Tópicos 309
Lambda Annotations 309
Bibliotecas e estruturas de alto nível 311
Estrutura de processamento de mensagens 311
Integração AWS com o.NET Aspire 332
Outros serviços e configuração 333
Exemplos de código 334
Aurora 335
Conceitos básicos 335
Conceitos básicos 337
Ações 362
ajuste de escala automático 377
Conceitos básicos 335
Conceitos básicos 337
Ações 362
Amazon Bedrock 413
Conceitos básicos 335
Ações 362
Amazon Bedrock Runtime 417
Claude da Anthropic 417

Command da Cohere ... 423
Llama da Meta ... 426
Mistral AI ... 430
CloudFormation ... 434
Conceitos básicos ... 335
CloudWatch ... 437
Conceitos básicos ... 335
Conceitos básicos ... 337
Ações ... 362
CloudWatch Registros ... 495
Ações ... 362
Cenários ... 499
Provedor de identidade do Amazon Cognito ... 521
Ações ... 362
AWS Control Tower ... 522
Conceitos básicos ... 335
Conceitos básicos ... 337
Ações ... 362
DynamoDB ... 561
Conceitos básicos ... 335
Conceitos básicos ... 337
Ações ... 362
Amazon EC2 ... 602
Conceitos básicos ... 335
Amazon ECS ... 604
Conceitos básicos ... 335
AWS IoT ... 606
Conceitos básicos ... 335
Conceitos básicos ... 337
Ações ... 362
AWS IoT data ... 665
Ações ... 362
banco de dados de origem ... 668
Conceitos básicos ... 335
Conceitos básicos ... 337
Ações ... 362

Amazon S3 ... 699
Conceitos básicos ... 335
Conceitos básicos ... 337
Ações ... 362
Cenários ... 499
Migrando seu projeto ... 735
Migrando para a versão 4 ... 735
NET Framework ... 735
Tipos de valor ... 736
Coleções ... 736
AWS Security Token Service (STS) ... 737
Mudanças relacionadas a `ClientConfig` ... 737
As AWSSDK extensões. NETCore.Pacote de configuração NuGet ... 738
`CookieSigner` e `UrlSigner` ... 739
DateTime versus UTC DateTime ... 739
DateTime analisando ... 739
`ConvertFromUnixEpochSeconds` e `ConvertFromUnixEpochMilliseconds` ... 740
Registro em log ... 740
Support para HTTP 2 ... 741
Autenticação única ... 741
Alterações específicas do DynamoDB ... 741
Alterações específicas do EC2 ... 745
Alterações específicas do S3 ... 746
Elementos de programação que foram removidos ... 749
Migração do .NET Standard 1.3 ... 752
Segurança ... 754
Proteção de dados ... 755
Gerenciamento de Identidade e Acesso ... 756
Público ... 756
Autenticação com identidades ... 757
Gerenciar o acesso usando políticas ... 758
Como Serviços da AWS trabalhar com o IAM ... 760
Solução de problemas AWS de identidade e acesso ... 760
Validação de conformidade ... 762
Resiliência ... 763
Segurança da infraestrutura ... 763

Aplicar uma versão mínima do TLS ... 764
.NET Core ... 764
NET Framework ... 765
Ferramentas da AWS para PowerShell ... 766
Xamarin ... 767
Unity ... 768
Navegador (para Blazor) WebAssembly ... 768
Migração do cliente de criptografia S3 (V1 para V2) ... 768
Visão geral da migração ... 769
Atualize os clientes existentes para clientes V1 de transição para ler novos formatos. ... 769
Migre clientes V1 de transição para clientes V2 para escrever novos formatos. ... 770
Atualize os clientes V2 para que não leiam mais os formatos V1. ... 774
Migração do cliente de criptografia S3 (V2 para V4) ... 774
Compreendendo os conceitos da V4 ... 775
Atualize os clientes existentes para ler os formatos V4 ... 776
Migre para clientes V4 ... 777
Exemplos adicionais de configuração ... 781
Considerações especiais ... 784
Obtenção de AWSSDK montagens ... 784
Faça download e extraia os arquivos ZIP ... 784
Acessar credenciais e perfis em um aplicativo ... 785
Exemplos para a aula CredentialProfileStoreChain ... 786
Exemplos de aulas SharedCredentialsFile e AWSCredentials fábrica ... 787
Suporte ao Unity ... 788
Suporte do Xamarin ... 789
Referência da API ... 790
Sobre as versões de referência da API ... 790
Histórico do documento ... 793

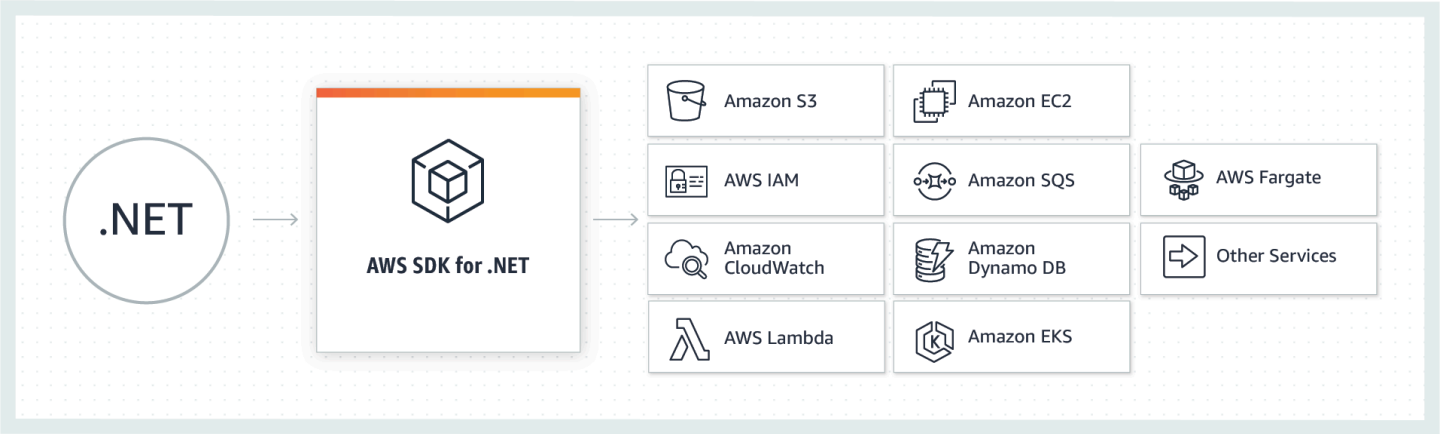
A versão 4 (V4) do AWS SDK para .NET foi lançada!

Para obter informações sobre mudanças significativas e migrar seus aplicativos, consulte o tópico de migração.

As traduções são geradas por tradução automática. Em caso de conflito entre o conteúdo da tradução e da versão original em inglês, a versão em inglês prevalecerá.

O que é o AWS SDK para .NET

AWS SDK para .NET Isso facilita a criação de aplicativos.NET que utilizam AWS serviços econômicos, escaláveis e confiáveis, como o Amazon Simple Storage Service (Amazon S3) e o Amazon Elastic Compute Cloud (Amazon). EC2 O SDK simplifica o uso dos serviços da AWS fornecendo um conjunto de bibliotecas que são consistentes e familiares para os desenvolvedores .NET.



(OK, entendi! Estou pronto para me preparar e fazer um tour rápido.)

Sobre essa versão

Documentação atual e anterior

Esta documentação é para a versão 4.0 e posterior (V4) do AWS SDK para .NET. Seu foco principal é o .NET Core e ASP.NET Core, mas ele também contém informações sobre o .NET Framework e ASP.NET 4.x. Além do Windows e do Visual Studio, ele considera o desenvolvimento multiplataforma igualmente.

Para obter informações sobre a migração de versões anteriores do SDK, consulte. Migrando seu projeto

Para obter informações sobre a versão anterior do SDK, V3, que será descontinuada no futuro, consulte AWS SDK para .NET o Guia do desenvolvedor (V3).

Para encontrar conteúdo obsoleto para versões anteriores do SDK, consulte o (s) seguinte (s) item (s):

- AWS SDK para .NET (versão 2, obsoleta) Guia do desenvolvedor

- Referências de API obsoletas para o AWS SDK para .NET

Manutenção e suporte para as versões principais do SDK

Para obter informações sobre manutenção e suporte para as versões principais do SDK e suas dependências subjacentes, consulte o seguinte no Guia de referência de ferramentas AWS SDKs e ferramentas:

- AWS SDKs e política de manutenção de ferramentas
- AWS SDKs matriz de suporte de versões e ferramentas

Casos de uso comuns

AWS SDK para .NET Isso ajuda você a realizar vários casos de uso convincentes, incluindo os seguintes:

- Gerenciar usuários e funções com AWS Identity and Access Management (IAM).
- Acessar o Amazon Simple Storage Service (Amazon S3) para criar buckets e armazenar objetos.
- Gerenciar assinaturas de tópicos de HTTP do Amazon Simple Notification Service (Amazon SNS).
- Usar o utilitário de transferência S3 para transferir arquivos de seus aplicativos Xamarin para o Amazon S3.
- Usar o Amazon Simple Queue Service (Amazon SQS) para processar mensagens e fluxos de trabalho entre componentes em um sistema.
- Executar transferências eficientes do Amazon S3 enviando instruções SQL para o Amazon S3 Select.
- Crie e execute EC2 instâncias da Amazon e configure e solicite instâncias EC2 spot da Amazon.

Tópicos adicionais nesta seção

- AWS ferramentas relacionadas ao AWS SDK para .NET
- Plataformas suportadas pelo AWS SDK para .NET
- AWS SDKs Guia de referência de ferramentas e ferramentas

- Histórico de revisões
- O que há de novo no AWS SDK para .NET
- Recursos adicionais

AWS ferramentas relacionadas ao AWS SDK para .NET

Ferramentas para Windows PowerShell e Ferramentas para PowerShell Core

Os AWS Tools for Windows PowerShell e AWS Tools for PowerShell Core são PowerShell módulos baseados na funcionalidade exposta pelo AWS SDK para .NET. As AWS PowerShell ferramentas permitem que você crie scripts de operações em seus AWS recursos a partir do PowerShell prompt. Embora os cmdlets sejam implementados usando os clientes de serviço e os métodos do SDK, eles fornecem uma PowerShell experiência idiomática para especificar parâmetros e lidar com resultados.

Para começar, consulte o AWS Tools for Windows PowerShell.

Toolkit for VS Code

O AWS Toolkit for Visual Studio Code é um plug-in para o editor Visual Studio Code (VS Code). O toolkit facilita o desenvolvimento, a depuração e a implantação de aplicativos que usam a AWS.

Com o toolkit, é possível fazer o seguinte:

- Crie aplicativos sem servidor que contenham AWS Lambda funções e, em seguida, implante os aplicativos em uma AWS CloudFormation pilha.
- Trabalhe com EventBridge esquemas da Amazon.
- Use IntelliSense ao trabalhar com arquivos de definição de tarefas do Amazon ECS.
- Visualize um AWS Cloud Development Kit (AWS CDK) aplicativo.

Toolkit para Visual Studio

AWS Toolkit for Visual Studio É um plug-in para o Visual Studio IDE que facilita o desenvolvimento, a depuração e a implantação de aplicativos.NET que usam a Amazon Web Services. O Toolkit para Visual Studio fornece modelos do Visual Studio para serviços, como o Lambda, e assistentes

de implantação para aplicativos web e aplicativos sem servidor. Você pode usar o AWS Explorer para gerenciar EC2 instâncias da Amazon, trabalhar com tabelas do Amazon DynamoDB, publicar mensagens nas filas do Amazon Simple Notification Service (Amazon SNS) e muito mais, tudo dentro do Visual Studio.

Para começar, consulte Setting up the AWS Toolkit for Visual Studio.

Kit de ferramentas para Azure DevOps

O Kit de ferramentas da AWS para Microsoft Azure DevOps adiciona tarefas para habilitar facilmente pipelines de criação e lançamento no Azure DevOps e no Azure DevOps Server para trabalhar com AWS serviços. Você pode trabalhar com Amazon S3,,, AWS CodeDeploy Lambda AWS Elastic Beanstalk AWS CloudFormation, Amazon Simple Queue Service (Amazon SQS) Amazon SQS e Amazon SNS. Você também pode executar comandos usando o PowerShell módulo Windows e o AWS Command Line Interface (AWS CLI).

Para começar a usar o Kit de ferramentas da AWS para Azure DevOps, consulte o Guia Kit de ferramentas da AWS para Microsoft Azure DevOps do usuário.

Plataformas suportadas pelo AWS SDK para .NET

O AWS SDK para .NET fornece grupos distintos de assemblies para que os desenvolvedores tenham como alvo plataformas diferentes. Contudo, nem toda funcionalidade do SDK é a mesma em cada uma dessas plataformas. Este tópico descreve as diferenças no suporte para cada plataforma.

.NET Core

O AWS SDK para .NET suporta aplicativos escritos para o.NET Core (.NET Core 3.1, .NET 5, .NET 6 e assim por diante). AWS os clientes de serviço oferecem suporte somente a padrões de chamadas assíncronas no.NET core. Isso também afeta várias das abstrações de alto nível criadas sobre clientes de serviços, como o `TransferUtility` do Amazon S3, que só oferece suporte a chamadas assíncronas no ambiente .NET Core.

.NET Standard 2.0

Variações não estruturais do AWS SDK para .NET estão em conformidade com o.NET Standard 2.0. O AWS SDK para .NET fornece somente métodos assíncronos para aplicativos escritos com o.NET Standard.

.NET Framework 4.5

Essa versão do AWS SDK para .NET é compilada com o.NET Framework 4.7.2 e é executada no tempo de execução do.NET 4.0. AWS os clientes de serviço oferecem suporte a padrões de chamada síncrona e assíncrona e usam as palavras-chave async e await introduzidas no C# 5.0.

.NET Framework 3.5

A versão 4 do AWS SDK para .NET não é compatível com o.NET Framework 3.5.

Biblioteca de classes portátil e Xamarin

AWS SDK para .NET Também contém uma implementação da Portable Class Library. A implementação da biblioteca de classes portátil pode ser destinada a várias plataformas, incluindo Universal Windows Platform (UWP) e Xamarin em iOS e Android. Consulte o SDK móvel para .NET e Xamarin para obter mais detalhes. AWS os clientes de serviço oferecem suporte somente a padrões de chamadas assíncronas.

Suporte ao Unity

Para obter informações sobre o suporte do Unity, consulte Considerações especiais sobre o suporte ao Unity.

AWS SDKs Guia de referência de ferramentas e ferramentas

O Guia de Referência de Ferramentas AWS SDKs e Ferramentas contém informações que são relevantes e importantes para muitos dos kits de ferramentas AWS SDKs e o. AWS CLI Veja a seguir alguns exemplos das informações que a referência contém:

- Informações sobre os arquivos compartilhados `config` e `credentials` da AWS e a localização deles.
- Configurando AWS contas, usuários e funções
- Referência de configurações e definições de autenticação
- AWS Bibliotecas do Common Runtime (CRT)
- AWS SDKs e política de manutenção de ferramentas
- AWS SDKs matriz de suporte de versões e ferramentas

Histórico de revisões

Para descobrir o que mudou em várias versões, veja o seguinte:

- Registros de alterações do SDK
- O que há de novo no AWS SDK para .NET
- Histórico do documento

O que há de novo no AWS SDK para .NET

Para obter informações de alto nível sobre novos desenvolvimentos relacionados ao AWS SDK para .NET, consulte a página do produto em https://aws.amazon.com/sdk-for-net/e os registros de alterações do SDK.

Veja a seguir o que há de novo no AWS SDK para .NET.

17 de outubro de 2025: Versão 2.0 da ferramenta AWS Deploy

A versão 2.0 da ferramenta AWS Deploy Tool for .NET CLI foi lançada. Para obter informações sobre a ferramenta de implantação, consulteImplemente aplicativos em AWS. Para obter informações sobre a versão 2.0 da ferramenta de implantação, consulte a postagem do blog O que há de novo na ferramenta de AWS implantação para.NET.

21 de agosto de 2025: Disponível end-of-support para a versão 3 do AWS SDK para .NET

A end-of-support versão 3 (V3) do AWS SDK para .NET foi anunciada. Consulte o guia de migração para migrar seus aplicativos V3 e evitar interrupções. Para obter mais informações, consulte a postagem do blog Anunciando o end-of-support para a AWS SDK para .NET v3.

2 de julho de 2025: disponibilidade geral do provedor de cache distribuído

O provedor de cache distribuído AWS do.NET para o Amazon DynamoDB está disponível ao público em geral! Esse provedor implementa a interface ASP.NET Core IDistributedCache, permitindo que você integre a infraestrutura totalmente gerenciada e durável do DynamoDB à sua camada de cache com o mínimo de alterações no código. O provedor de cache distribuído está disponível por meio da AWS. AspNetCore. DistributedCacheProviderpacote ligado NuGet. Para obter mais informações, consulte a postagem do blog Provedor de cache distribuído AWS do.NET para Amazon DynamoDB agora disponível ao público em geral.

> **Note**
>
> A versão lançada do Distributed Cache Provider é suportada somente na V4 do AWS SDK para .NET. Se você experimentou a versão prévia do Distributed Cache Provider, talvez seja necessário atualizar as dependências do pacote em seu aplicativo.

5 de maio de 2025: Disponibilidade geral da Estrutura de Processamento de AWS Mensagens para.NET

O AWS Message Processing Framework para.NET está disponível ao público em geral! A estrutura é AWS nativa que simplifica o desenvolvimento de aplicativos de processamento de mensagens.NET que usam AWS serviços como Amazon Simple Queue Service (SQS), Amazon Simple Notification Service (SNS) e Amazon. EventBridge

28 de abril de 2025: Versão 4 do AWS SDK para .NET

A versão 4 do AWS SDK para .NET está geralmente disponível! Para obter informações sobre como migrar seus aplicativos para a V4, consulte. Migrando para a versão 4 Veja também a postagem do blog Disponibilidade geral da AWS SDK para .NET V4.0 que anuncia a disponibilidade geral.

15 de fevereiro de 2025: Integrações com o.NET Aspire

Foram lançadas integrações com o.NET Aspire para melhorar o ciclo interno de desenvolvimento. Para mais informações, consulte Integração AWS com o.NET Aspire no AWS SDK para .NET.

10 de fevereiro de 2025: versão GA para observabilidade

Observabilidade é até que ponto o estado atual de um sistema pode ser inferido a partir dos dados que ele emite. A observabilidade foi adicionada ao AWS SDK para .NET, incluindo a implementação de um provedor de telemetria. Para obter mais informações, consulte Observabilidade este guia e a postagem do blog Anunciando a disponibilidade geral das AWS OpenTelemetry bibliotecas.NET.

15 de janeiro de 2025: novo comportamento padrão para proteção de integridade

A partir da versão 3.7.412.0 do AWS SDK para .NET, o SDK fornece proteções de integridade padrão calculando automaticamente uma soma de verificação para uploads. `CRC32` Para obter mais informações, consulte o anúncio GitHub emhttps://github.com/aws/aws-sdk-net/issues/3610. O SDK também fornece configurações globais para proteções de integridade de dados que você pode definir

externamente, sobre as quais você pode ler em Proteções de integridade de dados no Guia de referência de ferramentas AWS SDKs e ferramentas.

15 de novembro de 2024: publicação no blog da versão prévia 4 da versão 4

A prévia 4 da AWS SDK para .NET versão 4 foi lançada em 15 de novembro de 2024. Para obter mais informações sobre essa prévia, consulte a postagem do blog Preview 4 of AWS SDK para .NET V4.

16 de agosto de 2024: publicação no blog da versão prévia 1 da versão 4

A AWS SDK para .NET versão 4 foi lançada como primeira prévia em 16 de agosto de 2024. Para obter mais informações sobre essa prévia, consulte a postagem do blog Prévia 1 da AWS SDK para .NET V4.

Recursos adicionais

Serviços com suporte

AWS SDK para .NET Ele suporta a maioria dos produtos de AWS infraestrutura e mais serviços são adicionados com frequência. Para ver uma lista dos AWS serviços compatíveis com o SDK, consulte o arquivo README do SDK.

Página inicial do AWS SDK para .NET

Para obter mais informações sobre o AWS SDK para .NET, consulte a página inicial do SDK em https://aws.amazon.com/sdk-for-net/.

Documentação de referência do SDK

A documentação de referência do SDK dá a você a capacidade de pesquisar e buscar todos os códigos inclusos no SDK. Ele fornece documentação completa e exemplos de uso. Para obter mais informações, consulte a Referência da API do AWS SDK para .NET.

AWS re:post (antigos AWS fóruns)

Visite AWS re:POST, especificamente o tópico do AWS SDK para .NET, para fazer perguntas ou fornecer feedback sobre. AWS Cada página de documentação tem um link Experimente o AWS re:post na parte inferior da página que leva você ao tópico re:POST associado. Engenheiros da AWS monitoram os tópicos e respondem a perguntas, feedback e problemas.

Se você estiver conectado ao re:POST, você também pode seguir um tópico. Para seguir o tópico do AWS SDK para .NET, acesse a página Todos os tópicos, localize ".NET em AWS"e selecione o botão Seguir.

Toolkits

- AWS Toolkit for Visual Studio: Se usar o Microsoft Visual Studio IDE, você deverá verificar o Guia do usuário do AWS Toolkit for Visual Studio.
- AWS Toolkit for Visual Studio Code: Se usar o Microsoft Visual Studio IDE, você deverá verificar o Guia do usuário do AWS Toolkit for Visual Studio Code.

Extensões, bibliotecas e ferramentas úteis

Visite os aws-sdk-net repositórios aws/dotnet e aws/ no GitHub site para obter links para bibliotecas, ferramentas e recursos que você pode usar para ajudar a criar aplicativos e serviços do.NET. AWS

Veja os seguintes exemplos:

- AWS Extensão de configuração.NET para Systems Manager
- AWS Configuração do Extensions .NET Core
- AWS Registrando o.NET
- Biblioteca de extensão de autenticação do Amazon Cognito
- AWS X-Ray SDK for .NET

Outros recursos

A seguir estão outros recursos que podem ser úteis:

- Rede de desenvolvedores
- Ambiente de desenvolvimento.NET na AWS nuvem - Implantação de referência de início rápido
- Blog Hello, Cloud!
- AWS Documento técnico: Desenvolvendo e implantando aplicativos.NET em AWS
- AWS Microservice Extractor for .NET
- Assistente de portabilidade para .NET
- AWS SDKs Guia de referência de ferramentas e ferramentas

Começando com o AWS SDK para .NET

Saiba como instalar, configurar e usar AWS SDK para .NET o. Você deve instalar seu conjunto de ferramentas e configurar várias coisas essenciais que seu aplicativo precisa para acessar os AWS serviços. Isso inclui:

- Uma conta ou perfil de usuário apropriado
- Informações de autenticação para essa conta de usuário ou para assumir esse perfil

Para obter informações sobre como configurar projetos e outras partes críticas, consulteConfigurando o AWS SDK para .NET.

Tópicos

- Instalando e configurando seu conjunto de ferramentas para o AWS SDK para .NET
- Autenticando o com AWS SDK para .NET AWS
- Criando um aplicativo simples usando o AWS SDK para .NET

Instalando e configurando seu conjunto de ferramentas para o AWS SDK para .NET

Para usar o AWS SDK para .NET, você deve ter determinadas ferramentas de desenvolvimento instaladas.

Desenvolvimento entre plataformas

Os itens a seguir são necessários para desenvolvimento .NET para várias plataformas no Windows, no Linux ou no macOS:

- Microsoft .NET Core SDK, versão 2.1, 3.1 ou posterior, que inclua a interface de linha de comando (CLI) do .NET (**`dotnet`**) e o tempo de execução do .NET Core.

- Um editor de código ou ambiente de desenvolvimento integrado (IDE) apropriado para seu sistema operacional e requisitos. Normalmente, ele fornece algum suporte para .NET Core.

Os exemplos incluem Microsoft Visual Studio Code (VS Code), JetBrains Rider e Microsoft Visual Studio.

- (Opcional) Um AWS kit de ferramentas, se estiver disponível, para o editor que você escolheu e seu sistema operacional.

  Os exemplos incluem AWS Toolkit for Visual Studio Code, AWS Toolkit for JetBrains e AWS Toolkit for Visual Studio.

Windows com Visual Studio e .NET Core

Os itens a seguir são necessários para desenvolvimento no Windows com Visual Studio e .NET Core:

- Microsoft Visual Studio

- Microsoft .NET Core 2.1, 3.1 ou posterior

  Isso geralmente é incluído por padrão ao instalar uma versão recente do Visual Studio.

- (Opcional) O AWS Toolkit for Visual Studio, que é um plug-in que fornece uma interface de usuário para gerenciar seus AWS recursos e perfis locais do Visual Studio. Para instalar o toolkit, consulte Setting up the AWS Toolkit for Visual Studio.

  Para obter mais informações, consulte o Guia do usuário do AWS Toolkit for Visual Studio.

Próxima etapa

Autenticando o com AWS SDK para .NET AWS

Autenticando o com AWS SDK para .NET AWS

Você deve estabelecer como seu código é autenticado AWS ao desenvolver com Serviços da AWS. Há diferentes maneiras de configurar o acesso programático aos AWS recursos, dependendo do ambiente e do AWS acesso disponível para você.

Para ver vários métodos de autenticação para o SDK, consulte Autenticação e acesso no Guia de referência de ferramentas AWS SDKs e ferramentas.

Este tópico pressupõe que um novo usuário esteja se desenvolvendo localmente, não tenha recebido um método de autenticação do empregador e o usará Centro de Identidade do AWS IAM para obter credenciais temporárias. Se seu ambiente não se enquadra nessas suposições, algumas das informações neste tópico podem não se aplicar a você ou algumas das informações podem já ter sido fornecidas.

A configuração desse ambiente requer várias etapas, que são resumidas da seguinte forma:

1. Faça login usando as credenciais do console
2. Habilitar e configurar o Centro de Identidade do IAM
3. Configure o SDK para usar o Centro de Identidade do IAM.
4. Iniciar uma sessão do portal de AWS acesso

Faça login usando as credenciais do console

Você pode usar suas credenciais de login existentes do AWS Management Console para acesso programático aos serviços. AWS Depois de um fluxo de autenticação baseado em navegador, AWS gera credenciais temporárias que funcionam em ferramentas de desenvolvimento locais, como a AWS CLI e o Ferramentas da AWS para PowerShell SDK for AWS .NET v4.

Com esse processo, você pode se autenticar usando credenciais raiz criadas durante a configuração inicial da conta, um usuário do IAM ou uma identidade federada do seu provedor de identidade, e a AWS CLI ou gerencia Ferramentas da AWS para PowerShell automaticamente as credenciais temporárias para você. Essa abordagem aprimora a segurança ao eliminar a necessidade de armazenar credenciais de longo prazo localmente. Ao executar o `Invoke-AWSLogin` cmdlet Ferramentas da AWS para PowerShell ou o `aws login` comando da AWS CLI, você pode selecionar entre suas sessões ativas do console ou entrar por meio do fluxo de autenticação baseado em navegador e isso gerará automaticamente credenciais temporárias. O token de credencial temporário expirará em 15 minutos, mas o AWS SDK para.NET v4 atualiza automaticamente o token quando necessário durante suas solicitações.

> **Important**
>
> Se você estiver usando as credenciais do console para autenticação, seu aplicativo deverá fazer referência aos seguintes NuGet pacotes para que a resolução de AWS login funcione:

- AWSSDK.Iniciar sessão

A falha em referenciar esses pacotes resultará em uma exceção de runtime.

- Ferramentas da AWS para PowerShell Guia do desenvolvedor de credenciais de login.
- Faça login para desenvolvimento AWS local usando credenciais de console usando AWS CLI.

O `config` arquivo ficaria assim que você fizesse o login seguindo as etapas fornecidas acima para Ferramentas da AWS para PowerShell nossa AWS CLI.

```
[default]
login_session = arn:aws:iam::0123456789012:user/username
region = us-east-1

[my-dev-profile]
login_session = arn:aws:iam::0123456789012:user/username
region = us-east-1
```

Habilitar e configurar o Centro de Identidade do IAM

Para usar o Centro de Identidade do IAM, primeiro ele deve ser habilitado e configurado. Para ver detalhes sobre como fazer isso com o SDK, consulte a Etapa 1 no tópico sobre autenticação do IAM Identity Center no Guia de referência de ferramentas AWS SDKs e ferramentas. Especificamente, siga todas as instruções necessárias em Não estabeleci acesso por meio do Centro de Identidade do IAM.

Configure o SDK para usar o Centro de Identidade do IAM.

As informações sobre como configurar o SDK para usar o IAM Identity Center estão na Etapa 2 do tópico sobre autenticação do IAM Identity Center no Guia de referência de ferramentas AWS SDKs e ferramentas. Depois de concluir essa configuração, o sistema deverá conter os seguintes elementos:

- O AWS CLI, que você usa para iniciar uma sessão do portal de AWS acesso antes de executar seu aplicativo.
- O AWS `config` arquivo compartilhado que contém um `[default]`perfil com um conjunto de valores de configuração que podem ser referenciados no SDK. Para encontrar a localização desse

arquivo, consulte Localização dos arquivos compartilhados no Guia de referência de ferramentas AWS SDKs e ferramentas. O AWS SDK para .NET usa o provedor de token SSO do perfil para adquirir credenciais antes de enviar solicitações para. AWS O valor `sso_role_name`, que é um perfil do IAM conectado a um conjunto de permissões do Centro de Identidade do IAM, deve permitir o acesso a Serviços da AWS usado na aplicação.

O arquivo de exemplo `config` a seguir mostra um perfil padrão configurado com o provedor de token de SSO. A configuração `sso_session` do perfil se refere à seção chamada `sso-session`. A `sso-session` seção contém configurações para iniciar uma sessão do portal de AWS acesso.

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

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

Important

Se você estiver usando Centro de Identidade do AWS IAM para autenticação, seu aplicativo deverá fazer referência aos seguintes NuGet pacotes para que a resolução de SSO funcione:

- `AWSSDK.SSO`
- `AWSSDK.SSOOIDC`

A falha em referenciar esses pacotes resultará em uma exceção de runtime.

Iniciar uma sessão do portal de AWS acesso

Antes de executar um aplicativo que acessa Serviços da AWS, você precisa de uma sessão ativa do portal de AWS acesso para que o SDK use a autenticação do IAM Identity Center para resolver as credenciais. Dependendo da duração da sessão configurada, o seu acesso acabará expirando e o

SDK encontrará um erro de autenticação. Para entrar no portal de AWS acesso, execute o seguinte comando no AWS CLI.

```
aws sso login
```

Como você tem uma configuração de perfil padrão, não precisa chamar o comando com uma opção `--profile`. Se a configuração do provedor de token de SSO estiver usando um perfil nomeado, o comando será `aws sso login --profile named-profile`.

Para testar se você já tem uma sessão ativa, execute o AWS CLI comando a seguir.

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

A resposta a esse comando deve relatar a conta do Centro de Identidade do IAM e o conjunto de permissões configurados no arquivo compartilhado `config`.

> **Note**
>
> Se você já tiver uma sessão ativa do portal de AWS acesso e executá-la`aws sso login`, não será necessário fornecer credenciais.
> O processo de login pode solicitar que você permita o AWS CLI acesso aos seus dados.
> Como o AWS CLI é construído sobre o SDK para Python, as mensagens de permissão podem conter variações do `botocore` nome.

Mais informações

- Para obter informações adicionais sobre como usar o IAM Identity Center e o SSO em um ambiente de desenvolvimento, consulte Autenticação única na seção Autenticação avançada. Essas informações incluem métodos alternativos e mais avançados, bem como tutoriais que mostram como usar esses métodos.
- Para obter mais opções de autenticação para o SDK, como o uso de perfis e variáveis de ambiente, consulte o capítulo de configuração no Guia de referência de ferramentas AWS SDKs e ferramentas.
- Para saber mais sobre as práticas recomendadas, consulte Práticas recomendadas de segurança no IAM no Guia do usuário do IAM.

- Para criar AWS credenciais de curto prazo, consulte Credenciais de segurança temporárias no Guia do usuário do IAM.
- Para saber mais sobre outros provedores de credenciais, consulte Provedores de credenciais padronizados no Guia de Referência de Ferramentas AWS SDKs e Ferramentas.

Criando um aplicativo simples usando o AWS SDK para .NET

Essa seção fornece tutoriais básicos para desenvolvedores que não estão familiarizados com o AWS SDK para .NET.

> **Note**
>
> Antes de usar esses tutoriais, primeiro você deve ter instalado seu conjunto de ferramentas e configurado a autenticação do SDK.

Para obter informações sobre o desenvolvimento de software para AWS serviços específicos, juntamente com exemplos de código, consulte AWS Serviços de chamadas. Consulte mais exemplos de código em SDK para .NET (v4) exemplos de código.

Tópicos

- Aplicativo multiplataforma simples usando o AWS SDK para .NET
- Aplicativo simples baseado em Windows usando o AWS SDK para .NET
- Próximas etapas

Aplicativo multiplataforma simples usando o AWS SDK para .NET

Este tutorial usa o.NET Core para desenvolvimento multiplataforma. AWS SDK para .NET O tutorial mostra como usar o SDK para listar os buckets do Amazon S3que você possui e criar um bucket de forma opcional.

Você executará este tutorial usando ferramentas para várias plataforma, como a interface de linha de comando (CLI) do .NET. Para outras maneiras de configurar seu ambiente de desenvolvimento, consulte Instalando e configurando seu conjunto de ferramentas para o AWS SDK para .NET.

Necessário para desenvolvimento .NET para várias plataformas no Windows, no Linux ou no macOS:

- Microsoft .NET Core SDK, versão 2.1, 3.1 ou posterior, que inclua a interface de linha de comando (CLI) do .NET (**`dotnet`**) e o tempo de execução do .NET Core.

- Um editor de código ou ambiente de desenvolvimento integrado (IDE) apropriado para seu sistema operacional e requisitos. Normalmente, ele fornece algum suporte para .NET Core.

  Os exemplos incluem Microsoft Visual Studio Code (VS Code), JetBrains Rider e Microsoft Visual Studio.

> **Note**
>
> Antes de usar esses tutoriais, primeiro você deve ter instalado seu conjunto de ferramentas e configurado a autenticação do SDK.

Etapas

- Criar o projeto
- Criar o código
- Execute o aplicativo
- Limpeza

Criar o projeto

1. Abra o prompt de comando ou terminal. Localize ou crie uma pasta do sistema operacional na qual você pode criar um projeto .NET.
2. Nessa pasta, execute o comando a seguir para criar o projeto .NET.

```
dotnet new console --name S3CreateAndList
```

3. Vá para a pasta `S3CreateAndList` recém-criada e execute os comandos a seguir.

```
dotnet add package AWSSDK.S3
dotnet add package AWSSDK.SecurityToken
dotnet add package AWSSDK.SSO
dotnet add package AWSSDK.SSOOIDC
```

Os comandos anteriores instalam os NuGet pacotes do gerenciador de NuGet pacotes. Como sabemos exatamente quais NuGet pacotes precisamos para este tutorial, podemos realizar essa etapa agora. Também é bastante comum conhecer os pacotes necessários durante o desenvolvimento. Quando isso acontece, um comando semelhante pode ser executado nesse momento.

Criar o código

1. Na pasta `S3CreateAndList`, localize e abra `Program.cs` no editor de código.
2. Substitua o conteúdo pelo código a seguir e salve o arquivo.

```
using System;
using System.Threading.Tasks;

// NuGet packages: AWSSDK.S3, AWSSDK.SecurityToken, AWSSDK.SSO, AWSSDK.SSOOIDC
using Amazon.Runtime;
using Amazon.Runtime.CredentialManagement;
using Amazon.S3;
using Amazon.S3.Model;
using Amazon.SecurityToken;
using Amazon.SecurityToken.Model;

namespace S3CreateAndList
{
    class Program
    {
        // This code is part of the quick tour in the developer guide.
        // See https://docs.aws.amazon.com/sdk-for-net/v3/developer-guide/quick-
start.html
        // for complete steps.
        // Requirements:
        // - An SSO profile in the SSO user's shared config file with sufficient
 privileges for
  //   STS and S3 buckets.
        // - An active SSO Token.
        //    If an active SSO token isn't available, the SSO user should do the
 following:
        //    In a terminal, the SSO user must call "aws sso login".

        // Class members.
```

```
        static async Task Main(string[] args)
        {
            // Get SSO credentials from the information in the shared config file.
            // For this tutorial, the information is in the [default] profile.
            var ssoCreds = LoadSsoCredentials("default");

            // Display the caller's identity.
            var ssoProfileClient = new AmazonSecurityTokenServiceClient(ssoCreds);
            Console.WriteLine($"\nSSO Profile:\n {await
ssoProfileClient.GetCallerIdentityArn()}");

            // Create the S3 client is by using the SSO credentials obtained
earlier.
            var s3Client = new AmazonS3Client(ssoCreds);

            // Parse the command line arguments for the bucket name.
            if (GetBucketName(args, out String bucketName))
            {
                // If a bucket name was supplied, create the bucket.
                // Call the API method directly
                try
                {
                    Console.WriteLine($"\nCreating bucket {bucketName}...");
                    var createResponse = await s3Client.PutBucketAsync(bucketName);
                    Console.WriteLine($"Result:
{createResponse.HttpStatusCode.ToString()}");
                }
                catch (Exception e)
                {
                    Console.WriteLine("Caught exception when creating a bucket:");
                    Console.WriteLine(e.Message);
                }
            }

            // Display a list of the account's S3 buckets.
            Console.WriteLine("\nGetting a list of your buckets...");
            var listResponse = await s3Client.ListBucketsAsync();
            Console.WriteLine($"Number of buckets: {listResponse.Buckets.Count}");
            foreach (S3Bucket b in listResponse.Buckets)
            {
                Console.WriteLine(b.BucketName);
            }
            Console.WriteLine();
        }
```

```
        //
        // Method to parse the command line.
        private static Boolean GetBucketName(string[] args, out String bucketName)
        {
            Boolean retval = false;
            bucketName = String.Empty;
            if (args.Length == 0)
            {
                Console.WriteLine("\nNo arguments specified. Will simply list your
Amazon S3 buckets." +
                  "\nIf you wish to create a bucket, supply a valid, globally
unique bucket name.");
                bucketName = String.Empty;
                retval = false;
            }
            else if (args.Length == 1)
            {
                bucketName = args[0];
                retval = true;
            }
            else
            {
                Console.WriteLine("\nToo many arguments specified." +
                  "\n\ndotnet_tutorials - A utility to list your Amazon S3 buckets
and optionally create a new one." +
                  "\n\nUsage: S3CreateAndList [bucket_name]" +
                  "\n - bucket_name: A valid, globally unique bucket name." +
                  "\n - If bucket_name isn't supplied, this utility simply lists
your buckets.");
                Environment.Exit(1);
            }
            return retval;
        }

        //
        // Method to get SSO credentials from the information in the shared config
file.
        static AWSCredentials LoadSsoCredentials(string profile)
        {
            var chain = new CredentialProfileStoreChain();
            if (!chain.TryGetAWSCredentials(profile, out var credentials))
                throw new Exception($"Failed to find the {profile} profile");
            return credentials;
```

```
        }
    }

    // Class to read the caller's identity.
    public static class Extensions
    {
        public static async Task<string> GetCallerIdentityArn(this
 IAmazonSecurityTokenService stsClient)
        {
            var response = await stsClient.GetCallerIdentityAsync(new
 GetCallerIdentityRequest());
            return response.Arn;
        }
    }
}
```

Execute o aplicativo

1. Execute o seguinte comando:

```
dotnet run
```

2. Examine a saída para ver o número de buckets do Amazon S3 que você possui e seus nomes, se houver.
3. Escolha um nome para um novo bucket do Amazon S3. Use "dotnet-quicktour-s3-1-cross-" como base e adicione algo exclusivo, como um GUID ou seu nome. Siga as regras para nomes de bucket descritas em Regras de nomeação de bucket no Guia do usuário do Amazon S3.
4. Execute o comando a seguir, *amzn-s3-demo-bucket* substituindo-o pelo nome do bucket que você escolheu.

```
dotnet run amzn-s3-demo-bucket
```

5. Examine a saída para ver o novo bucket que foi criado.

Limpeza

Ao executar este tutorial, você criou alguns recursos que pode escolher para limpar neste momento.

- Se você não quiser manter o bucket que o aplicativo criou em uma etapa anterior, exclua-o usando o console do Amazon S3 em. https://console.aws.amazon.com/s3/
- Se você não quiser manter seu projeto .NET, remova a pasta `S3CreateAndList` do seu ambiente de desenvolvimento.

Para onde ir em seguida

Volte para o menu do tour rápido ou vá direto para o final deste tour rápido.

Aplicativo simples baseado em Windows usando o AWS SDK para .NET

Este tutorial usa o AWS SDK para .NET no Windows com Visual Studio e o.NET Core. O tutorial mostra como usar o SDK para listar os buckets do Amazon S3 que você possui e criar um bucket de forma opcional.

Você executará este tutorial no Windows usando o Visual Studio e o .NET Core. Para outras maneiras de configurar seu ambiente de desenvolvimento, consulte Instalando e configurando seu conjunto de ferramentas para o AWS SDK para .NET.

Necessário para desenvolvimento no Windows com o Visual Studio e o .NET Core:

- Microsoft Visual Studio

- Microsoft .NET Core 2.1, 3.1 ou posterior

  Isso geralmente é incluído por padrão ao instalar uma versão recente do Visual Studio.

> **Note**
>
> Antes de usar esses tutoriais, primeiro você deve ter instalado seu conjunto de ferramentas e configurado a autenticação do SDK.

Etapas

- Criar o projeto
- Criar o código
- Execute o aplicativo

- Limpeza

Criar o projeto

1. Abra o Visual Studio e crie um novo projeto que use a versão C# do modelo de aplicativo de console; ou seja, com a descrição: "... para criar um aplicativo de linha de comando que pode ser executado no.NET...". Nomeie o projeto `S3CreateAndList`.

   > **Note**
   >
   > Não escolha a versão do.NET Framework do modelo de aplicativo de console ou, se escolher, certifique-se de usar o.NET Framework 4.7.2 ou posterior.

2. Com o projeto recém-criado carregado, escolha Tools, NuGetPackage Manager, Manage NuGet Packages for Solution.
3. Procure os seguintes NuGet pacotes e instale-os no projeto:`AWSSDK.S3`,`AWSSDK.SecurityToken`,`AWSSDK.SSO`, e `AWSSDK.SSOOIDC`

   Esse processo instala os NuGet pacotes do gerenciador de NuGet pacotes. Como sabemos exatamente quais NuGet pacotes precisamos para este tutorial, podemos realizar essa etapa agora. Também é bastante comum conhecer os pacotes necessários durante o desenvolvimento. Quando isso acontecer, siga um processo semelhante para instalá-los nesse momento.
4. Se você pretende executar o aplicativo no prompt de comando, abra um prompt de comando agora e navegue até a pasta que conterá a saída de compilação. Normalmente, isso é algo como `S3CreateAndList\S3CreateAndList\bin\Debug\net6.0`, mas dependerá do seu ambiente.

Criar o código

1. No projeto `S3CreateAndList`, localize e abra `Program.cs` no IDE.
2. Substitua o conteúdo pelo código a seguir e salve o arquivo.

```
using System;
using System.Threading.Tasks;

// NuGet packages: AWSSDK.S3, AWSSDK.SecurityToken, AWSSDK.SSO, AWSSDK.SSOOIDC
```

```
using Amazon.Runtime;
using Amazon.Runtime.CredentialManagement;
using Amazon.S3;
using Amazon.S3.Model;
using Amazon.SecurityToken;
using Amazon.SecurityToken.Model;

namespace S3CreateAndList
{
    class Program
    {
        // This code is part of the quick tour in the developer guide.
        // See https://docs.aws.amazon.com/sdk-for-net/v3/developer-guide/quick-
start.html
        // for complete steps.
        // Requirements:
        // - An SSO profile in the SSO user's shared config file with sufficient
 privileges for
  //   STS and S3 buckets.
        // - An active SSO Token.
        //    If an active SSO token isn't available, the SSO user should do the
 following:
        //    In a terminal, the SSO user must call "aws sso login".

        // Class members.
        static async Task Main(string[] args)
        {
            // Get SSO credentials from the information in the shared config file.
            // For this tutorial, the information is in the [default] profile.
            var ssoCreds = LoadSsoCredentials("default");

            // Display the caller's identity.
            var ssoProfileClient = new AmazonSecurityTokenServiceClient(ssoCreds);
            Console.WriteLine($"\nSSO Profile:\n {await
 ssoProfileClient.GetCallerIdentityArn()}");

            // Create the S3 client is by using the SSO credentials obtained
 earlier.
            var s3Client = new AmazonS3Client(ssoCreds);

            // Parse the command line arguments for the bucket name.
            if (GetBucketName(args, out String bucketName))
            {
                // If a bucket name was supplied, create the bucket.
```

```
                // Call the API method directly
                try
                {
                    Console.WriteLine($"\nCreating bucket {bucketName}...");
                    var createResponse = await s3Client.PutBucketAsync(bucketName);
                    Console.WriteLine($"Result:
{createResponse.HttpStatusCode.ToString()}");
                }
                catch (Exception e)
                {
                    Console.WriteLine("Caught exception when creating a bucket:");
                    Console.WriteLine(e.Message);
                }
            }

            // Display a list of the account's S3 buckets.
            Console.WriteLine("\nGetting a list of your buckets...");
            var listResponse = await s3Client.ListBucketsAsync();
            Console.WriteLine($"Number of buckets: {listResponse.Buckets.Count}");
            foreach (S3Bucket b in listResponse.Buckets)
            {
                Console.WriteLine(b.BucketName);
            }
            Console.WriteLine();
        }

        //
        // Method to parse the command line.
        private static Boolean GetBucketName(string[] args, out String bucketName)
        {
            Boolean retval = false;
            bucketName = String.Empty;
            if (args.Length == 0)
            {
                Console.WriteLine("\nNo arguments specified. Will simply list your
Amazon S3 buckets." +
                  "\nIf you wish to create a bucket, supply a valid, globally
unique bucket name.");
                bucketName = String.Empty;
                retval = false;
            }
            else if (args.Length == 1)
            {
                bucketName = args[0];
```

```
                retval = true;
            }
            else
            {
                Console.WriteLine("\nToo many arguments specified." +
                  "\n\ndotnet_tutorials - A utility to list your Amazon S3 buckets
 and optionally create a new one." +
                  "\n\nUsage: S3CreateAndList [bucket_name]" +
                  "\n - bucket_name: A valid, globally unique bucket name." +
                  "\n - If bucket_name isn't supplied, this utility simply lists
 your buckets.");
                Environment.Exit(1);
            }
            return retval;
        }

        //
        // Method to get SSO credentials from the information in the shared config
 file.
        static AWSCredentials LoadSsoCredentials(string profile)
        {
            var chain = new CredentialProfileStoreChain();
            if (!chain.TryGetAWSCredentials(profile, out var credentials))
                throw new Exception($"Failed to find the {profile} profile");
            return credentials;
        }
    }

    // Class to read the caller's identity.
    public static class Extensions
    {
        public static async Task<string> GetCallerIdentityArn(this
 IAmazonSecurityTokenService stsClient)
        {
            var response = await stsClient.GetCallerIdentityAsync(new
 GetCallerIdentityRequest());
            return response.Arn;
        }
    }
}
```

3. Compile o aplicativo.

> **Note**
>
> Se estiver usando uma versão mais antiga do Visual Studio, você pode receber um erro de compilação semelhante ao seguinte:
> “O atributo 'async main' não está disponível no C# 7.0. Use a versão 7.1 ou superior da linguagem.”
> Se você receber esse erro, configure seu projeto para usar uma versão posterior da linguagem. Isso normalmente é feito nas propriedades do projeto, Build, Advanced.

Execute o aplicativo

1. Execute o aplicativo sem argumentos de linha de comando. Faça isso no prompt de comando (se você abriu um anteriormente) ou no IDE.
2. Examine a saída para ver o número de buckets do Amazon S3 que você possui e seus nomes, se houver.
3. Escolha um nome para um novo bucket do Amazon S3. Use "dotnet-quicktour-s3-1-winvs-” como base e adicione algo exclusivo a ela, como um GUID ou seu nome. Siga as regras para nomes de bucket descritas em Regras de nomeação de bucket no Guia do usuário do Amazon S3.
4. Execute o aplicativo novamente, desta vez fornecendo o nome do bucket.

   Na linha de comando, *amzn-s3-demo-bucket* substitua o comando a seguir pelo nome do bucket que você escolheu.

   ```
   S3CreateAndList amzn-s3-demo-bucket
   ```

   Ou, se você estiver executando o aplicativo no IDE, escolha Projeto, CreateAndList Propriedades do S3, Depurar e insira o nome do bucket lá.
5. Examine a saída para ver o novo bucket que foi criado.

Limpeza

Ao executar este tutorial, você criou alguns recursos que pode escolher para limpar neste momento.

- Se você não quiser manter o bucket que o aplicativo criou em uma etapa anterior, exclua-o usando o console do Amazon S3 em. https://console.aws.amazon.com/s3/
- Se você não quiser manter seu projeto .NET, remova a pasta `S3CreateAndList` do seu ambiente de desenvolvimento.

Para onde ir em seguida

Volte para o menu do tour rápido ou vá direto para o final deste tour rápido.

Próximas etapas

Certifique-se de limpar todos os recursos criados durante a execução desses tutoriais. Eles podem ser AWS recursos ou recursos em seu ambiente de desenvolvimento, como arquivos e pastas.

Agora que você visitou o AWS SDK para .NET, talvez queira começar seu projeto.

Configurando o AWS SDK para .NET

Para usar o AWS SDK para .NET, você precisa configurar seu ambiente e também configurar várias coisas essenciais que seu aplicativo precisa para acessar AWS os serviços. Isso inclui:

- Especificação da AWS região
- AWSSDK pacotes ou montagens

Alguns dos tópicos desta seção fornecem informações sobre como configurar esses itens essenciais.

Outros tópicos desta e de outras seções fornecem informações sobre formas avançadas de configurar o seu projeto.

Tópicos

- Iniciar um novo projeto
- Instale AWSSDK pacotes com NuGet
- Instale AWSSDK montagens sem NuGet
- Definindo a AWS região para o AWS SDK para .NET
- Resolução de perfil e credenciais
- Novas tentativas e tempos limite
- Observabilidade
- Informações adicionais sobre usuários e perfis
- Configuração avançada para seu AWS SDK para .NET projeto
- Uso de credenciais herdadas

Iniciar um novo projeto

Há várias técnicas que você pode usar para iniciar um novo projeto para acessar AWS os serviços. A seguir estão alguns desses métodos:

- Se você é iniciante no desenvolvimento do.NET AWS ou, pelo menos AWS SDK para .NET, no, você pode ver exemplos completos emCriando um aplicativo simples. Ele fornece uma introdução ao SDK.

- É possível iniciar um projeto básico usando a CLI do .NET. Para ver um exemplo disso, abra um prompt de comando ou terminal, crie uma pasta ou diretório e navegue até ele e, em seguida, insira o seguinte.

  ```
  dotnet new console --name [SOME-NAME]
  ```

  É criado um projeto vazio ao qual você pode adicionar código e NuGet pacotes. Para obter mais informações, consulte o Guia do .NET Core.

  Para ver uma lista de modelos de projeto, use o seguinte: `dotnet new --list`

- AWS Toolkit for Visual Studio Isso inclui modelos de projetos em C# para uma variedade de AWS serviços. Depois de instalar o toolkit no Visual Studio, você poderá acessar os modelos ao criar um projeto.

  Para ver isso, acesse Trabalhar com AWS serviços no Guia AWS Toolkit for Visual Studio do usuário. Diversos exemplos nessa seção criam projetos.

- Se você desenvolve com o Visual Studio no Windows, mas sem o AWS Toolkit for Visual Studio, use suas técnicas típicas para criar um novo projeto.

  Para ver um exemplo, abra o Visual Studio e selecione Arquivo, Novo, Projeto. Pesquise por ".net core" e escolha a versão C# do modelo Console App (.NET Core) ou WPF App (.NET Core). É criado um projeto vazio ao qual você pode adicionar código e NuGet pacotes.

Você pode encontrar alguns exemplos de como trabalhar com AWS serviços emExemplos de código guiado.

> **Important**
>
> Se você estiver usando Centro de Identidade do AWS IAM para autenticação, seu aplicativo deverá fazer referência aos seguintes NuGet pacotes para que a resolução de SSO funcione:
>
> - `AWSSDK.SSO`
> - `AWSSDK.SSOOIDC`

A falha em referenciar esses pacotes resultará em uma exceção de runtime.

Instale AWSSDK pacotes com NuGet

NuGetié um sistema de gerenciamento de pacotes para a plataforma do.NET. Com NuGet, você pode instalar os AWSSDKpacotes, bem como várias outras extensões, em seu projeto. Para obter informações adicionais, consulte o repositório aws/dotnet no site. GitHub

NuGet sempre tem as versões mais recentes dos AWSSDK pacotes, bem como as versões anteriores. NuGetestá ciente das dependências entre pacotes e instala todos os pacotes necessários automaticamente.

Warning

A lista de NuGet pacotes pode incluir um chamado simplesmente "AWSSDK" (sem identificador anexado). NÃO instale este NuGet pacote; ele é antigo e não deve ser usado para novos projetos.

Os pacotes instalados com NuGet são armazenados com seu projeto em vez de em um local central. Isso permite a instalação de versões do conjunto específicas para um certo aplicativo, sem criar problemas de compatibilidade para outros aplicativos. Para obter mais informações sobre NuGet, consulte a NuGet documentação.

Note

Se você não puder ou não tiver permissão para baixar e instalar NuGet pacotes por projeto, poderá obter os AWSSDK assemblies e armazená-los localmente (ou localmente).
Se isso se aplica a você e você ainda não obteve os AWSSDK conjuntos, consulte. Obtenção de AWSSDK montagens Para aprender a usar os assemblies armazenados localmente, consulte Instale AWSSDK montagens sem NuGet.

Usando NuGet a partir do prompt de comando ou terminal

1. Acesse os AWSSDK pacotes em NuGet e determine quais pacotes você precisa em seu projeto; por exemplo, AWSSDK.S3.
2. Copie o comando da CLI do .NET da página desse pacote, como mostrado no exemplo a seguir.

   ```
   dotnet add package AWSSDK.S3 --version 3.3.110.19
   ```

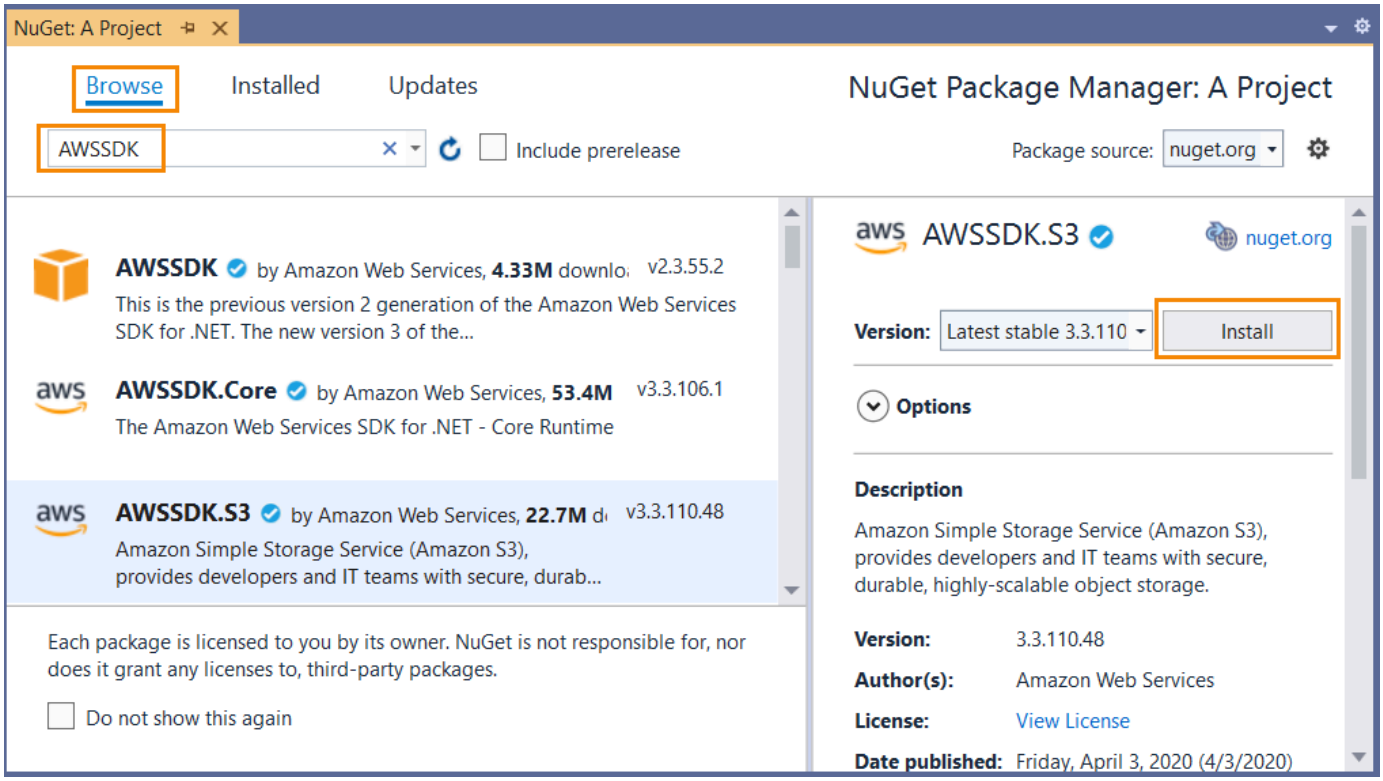
3. No diretório do seu projeto, execute esse comando.NET CLI. NuGet também instala quaisquer dependências, como AWSSDK .Core.

> **Note**
>
> Se você quiser somente a versão mais recente de um NuGet pacote, poderá excluir as informações da versão do comando, conforme mostrado no exemplo a seguir.
> ```
> dotnet add package AWSSDK.S3
> ```

Usando NuGet do Visual Studio Solution Explorer

1. No Solution Explorer, clique com o botão direito do mouse em seu projeto e escolha Gerenciar NuGet pacotes no menu de contexto.
2. No painel esquerdo do NuGet Package Manager, escolha Browse. Em seguida, você pode usar a caixa de pesquisa para pesquisar o pacote que deseja instalar. NuGet também instala quaisquer dependências, como AWSSDK .Core.

   A figura a seguir mostra a instalação do AWSSDKpacote.S3.



Usando NuGet a partir do Package Manager Console

No Visual Studio, escolha Tools, NuGet Package Manager, Package Manager Console.

Você pode instalar os AWSSDK pacotes desejados no Package Manager Console usando o **`Install-Package`**comando. Por exemplo, para instalar o AWSSDK.S3, use o comando a seguir.

```
PM> Install-Package AWSSDK.S3
```

NuGet também instala quaisquer dependências, como AWSSDK .Core.

Se você precisar instalar uma versão anterior de um pacote, use a opção `-Version` e especifique a versão do pacote que você deseja, conforme mostrado no exemplo a seguir.

```
PM> Install-Package AWSSDK.S3 -Version 3.3.106.6
```

Para obter mais informações sobre os comandos do Package Manager Console, consulte a PowerShellreferência na NuGetdocumentação da Microsoft.

Instale AWSSDK montagens sem NuGet

Este tópico descreve como você pode usar as AWSSDK montagens que você obteve e armazenou localmente (ou no local), conforme descrito em. Obtenção de AWSSDK montagens Esse não é o método recomendado para lidar com referências do SDK, mas é obrigatório em alguns ambientes.

> **Note**
>
> O método recomendado para lidar com referências do SDK é baixar e instalar apenas os NuGet pacotes de que cada projeto precisa. Esse método é descrito em Instale AWSSDK pacotes com NuGet.

Para instalar AWSSDK montagens

1. Crie uma pasta na área do projeto para as AWSSDK montagens necessárias. Por exemplo, você pode chamar essa pasta de `AwsAssemblies`.
2. Se você ainda não tiver feito isso, obtenha os AWSSDK assemblies, o que os coloca em alguma pasta local de download ou instalação. Copie os arquivos DLL dos assemblies necessários dessa pasta de download para o seu projeto (na pasta `AwsAssemblies`, em nosso exemplo).

   Assegure-se de também copiar as dependências. Você pode encontrar informações sobre dependências no GitHubsite.
3. Faça referência aos assemblies necessários da seguinte forma.

   Cross-platform development

   1. Abra o arquivo `.csproj` do seu projeto e adicione um elemento `<ItemGroup>`.
   2. No elemento `<ItemGroup>`, adicione um elemento `<Reference>` com um atributo `Include` para cada assembly necessário.

      Para o Amazon S3, por exemplo, você adicionaria as seguintes linhas ao arquivo `.csproj` do seu projeto.

      No Linux e macOS:

```
<ItemGroup>
  <Reference Include="./AwsAssemblies/AWSSDK.Core.dll" />
  <Reference Include="./AwsAssemblies/AWSSDK.S3.dll" />
```

```
</ItemGroup>
```

No Windows:

```
<ItemGroup>
  <Reference Include="AwsAssemblies\AWSSDK.Core.dll" />
  <Reference Include="AwsAssemblies\AWSSDK.S3.dll" />
</ItemGroup>
```

3. Salve o arquivo `.csproj` do seu projeto.

Windows with Visual Studio and .NET Core

1. No Visual Studio, carregue seu projeto e abra Projeto, Adicionar referência.
2. Escolha o botão Procurar na parte inferior da caixa de diálogo. Navegue até a pasta do seu projeto e a subpasta para a qual você copiou os arquivos DLL necessários (`AwsAssemblies`, por exemplo).
3. Selecione todos os arquivos DLL, escolha Adicionar e escolha OK.
4. Salve o seu projeto.

Definindo a AWS região para o AWS SDK para .NET

AWS As regiões permitem que você acesse AWS serviços que residem fisicamente em uma região geográfica específica. Isso pode ser útil para redundância e para manter os seus dados e aplicativos em execução próximo ao lugar em que você e os seus usuários os acessarão.

Para ver a lista atual de todas as regiões e endpoints compatíveis com cada AWS serviço, consulte Pontos de extremidade e cotas de serviço no. Referência geral da AWS Para ver uma lista dos endpoints regionais existentes, consulte endpoints de serviço da AWS. Para ver informações detalhadas sobre regiões, consulte Specify which AWS Regions your account can use.

Você pode criar um cliente AWS de serviço que vá para uma região específica. Você também pode configurar seu aplicativo com uma região que será usada para todos os clientes do AWS serviço. Esses dois casos serão explicados a seguir.

Crie um cliente de serviço com uma região específica

Você pode especificar a região para qualquer um dos clientes de AWS serviço em seu aplicativo. Definir a Região desta maneira tem precedência sobre qualquer configuração global para aquele cliente de serviço particular.

Região existente

Este exemplo mostra como instanciar um EC2 cliente da Amazon em uma região existente. Ele usa RegionEndpointcampos definidos.

```
using (AmazonEC2Client ec2Client = new AmazonEC2Client(RegionEndpoint.USWest2))
{
  // Make a request to EC2 in the us-west-2 Region using ec2Client
}
```

Nova região usando RegionEndpoint classe

Este exemplo mostra como construir um novo endpoint de região usandoRegionEndpoint. GetBySystemName.

```
var newRegion = RegionEndpoint.GetBySystemName("us-west-new");
using (var ec2Client = new AmazonEC2Client(newRegion))
{
  // Make a request to EC2 in the new Region using ec2Client
}
```

Nova região usando a classe de configuração do cliente de serviço

Este exemplo mostra como usar a `ServiceURL` propriedade da classe de configuração do cliente de serviço para especificar a região; nesse caso, usando a classe Amazon EC2 Config.

Esta técnica funciona mesmo se o endpoint da região não seguir o padrão regular de endpoints da região.

```
var ec2ClientConfig = new AmazonEC2Config
{
    // Specify the endpoint explicitly
    ServiceURL = "https://ec2.us-west-new.amazonaws.com"
};
```

```
using (var ec2Client = new AmazonEC2Client(ec2ClientConfig))
{
  // Make a request to EC2 in the new Region using ec2Client
}
```

Especifique uma região para todos os clientes de serviço

Há várias maneiras de especificar uma região para todos os clientes de AWS serviço que seu aplicativo cria. Essa região é usada para clientes de serviço que não foram criados com uma região específica.

O AWS SDK para .NET procura por um valor de região na seguinte ordem.

Perfis

Defina um perfil que seu aplicativo ou o SDK tenha carregado. Para obter mais informações, consulte Resolução de perfil e credenciais.

Variáveis de ambiente

Defina na variável de ambiente `AWS_REGION`.

No Linux ou macOS:

```
export AWS_REGION='us-west-2'
```

No Windows:

```
set AWS_REGION=us-west-2
```

> **Note**
>
> Se você definir essa variável de ambiente para todo o sistema (usando `export` ou`setx`), ela afetará todos SDKs os kits de ferramentas, não apenas o. AWS SDK para .NET

AWSConfigs classe

Definido como um AWSConfigs. AWSRegionpropriedade.

```
AWSConfigs.AWSRegion = "us-west-2";
using (var ec2Client = new AmazonEC2Client())
{
  // Make request to Amazon EC2 in us-west-2 Region using ec2Client
}
```

Resolução da região

Se nenhum dos métodos descritos acima for usado para especificar um Região da AWS, as AWS SDK para .NET tentativas de encontrar uma região para o cliente do AWS serviço operar.

Ordem da resolução da região

1. Arquivos de configuração do aplicativo, como `app.config` e `web.config`.
2. Variáveis de ambiente (`AWS_REGION` e `AWS_DEFAULT_REGION`).
3. Um perfil com o nome especificado por um valor em `AWSConfigs.AWSProfileName`.
4. Um perfil com o nome especificado pela variável de ambiente `AWS_PROFILE`.
5. O perfil `[default]`.
6. Metadados da EC2 instância Amazon (se estiver sendo executada em uma EC2 instância).

Se nenhuma região for encontrada, o SDK lançará uma exceção informando que o cliente do AWS serviço não tem uma região configurada.

Informações especiais sobre a região da China (Pequim)

Para usar serviços na região China (Pequim), é necessário ter uma conta e credenciais específicas para a região China (Pequim). Contas e credenciais de outras AWS regiões não funcionarão na região da China (Pequim). Da mesma forma, contas e credenciais da região da China (Pequim) não funcionarão para outras AWS regiões. Para ter informações sobre os endpoints e protocolos que estão disponíveis na região China (Pequim), consulte Endpoints da região Pequim.

Informações especiais sobre novos AWS serviços

Novos AWS serviços podem ser lançados inicialmente em algumas regiões e depois suportados em outras regiões. Nestes casos, não é necessário instalar o SDK mais recente para acessar as novas regiões para esse serviço. Você pode especificar regiões recém-adicionadas globalmente ou por cliente, como mostrado anteriormente.

Resolução de perfil e credenciais

Ele AWS SDK para .NET pesquisa as credenciais em uma determinada ordem e usa o primeiro conjunto disponível para o aplicativo atual.

Ordem de pesquisa de credencial

1. Credenciais definidas explicitamente no cliente do AWS serviço, conforme descrito em. Acessar credenciais e perfis em um aplicativo

   > **Note**
   >
   > Esse tópico está na seção Considerações especiais porque não é o método preferencial para especificar credenciais.

2. Sessões AWSCredentials criadas a partir das variáveis de `AWS_SESSION_TOKEN` ambiente `AWS_ACCESS_KEY_IDAWS_SECRET_ACCESS_KEY`, e, se todas as três variáveis tiverem um valor.
3. Básico AWSCredentials que é criado a partir das variáveis de `AWS_SECRET_ACCESS_KEY` ambiente `AWS_ACCESS_KEY_ID` e, se ambas as variáveis tiverem um valor.
4. AssumeRoleWithWebIdentityCredentialsque são criadas a partir das variáveis de `AWS_ROLE_ARN` ambiente `AWS_WEB_IDENTITY_TOKEN_FILE` e, se ambas as variáveis tiverem um valor.
5. Um perfil de credenciais com o nome especificado por um valor emAWSConfigs. AWSProfileNome.
6. Um perfil de credenciais com o nome especificado pela variável de ambiente `AWS_PROFILE`.
7. O perfil de credenciais `[default]`.
8. O provedor de credenciais do contêiner.
9. Metadados da EC2 instância Amazon.

Se seu aplicativo estiver sendo executado em uma EC2 instância da Amazon, como em um ambiente de produção, use uma função do IAM conforme descrito emConceder acesso utilizando um perfil do IAM. Caso contrário, como em testes de pré-lançamento, armazene suas credenciais em um arquivo que usa o formato de arquivo de AWS credenciais ao qual seu aplicativo web tem acesso no servidor.

Para obter informações adicionais sobre variáveis de ambiente e perfis de credenciais, consulte os seguintes tópicos no Guia de Referência de Ferramentas AWS SDKs e Ferramentas: Variáveis de ambiente, Lista de variáveis de ambiente e arquivos de configuração e credenciais compartilhados.

Resolução do perfil

Com dois mecanismos de armazenamento de credenciais diferentes, é importante entender como configurá-los AWS SDK para .NET para usá-los. AWSConfigsA. AWSProfilesA propriedade de localização controla como AWS SDK para .NET ele encontra perfis de credenciais.

AWSProfilesLocal	Comportamento da resolução do perfil
nulo (não definido) ou vazio	Pesquise na SDK Store se a plataforma for compatível e, em seguida, pesquise o arquivo de credenciais compartilhado da AWS no local padrão. Se o perfil não estiver em nenhum desses locais, busque `~/.aws/config` (Linux ou macOS) ou `%USERPROFILE%\.aws\config` (Windows).
O caminho para um arquivo no formato de arquivo de AWS credenciais	Pesquise somente o arquivo especificado para um perfil com o nome especificado.

Usar credenciais da conta de usuário federado

Os aplicativos que usam o AWS SDK para .NET (AWSSDK.Core versão 3.1.6.0 e posterior) podem usar contas de usuário federadas por meio dos Serviços de Federação do Active Directory (AD FS) para acessar AWS serviços usando o Security Assertion Markup Language (SAML).

Suporte ao acesso federado significa que usuários podem se autenticar usando o Active Directory. São concedidas credenciais temporárias ao usuário automaticamente. Essas credenciais temporárias, válidas por uma hora, são usadas quando seu aplicativo invoca serviços AWS . O SDK faz o gerenciamento das credenciais temporárias. Para contas de usuário associados a um domínio, se o aplicativo realizar uma chamada mas as credenciais expiraram, o usuário é reautenticado automaticamente e são concedidas novas credenciais. (Para contas que não estão associadas a um domínio, o usuário é solicitado a inserir as credenciais antes da reautenticação.)

Para usar esse suporte em seu aplicativo.NET, você deve primeiro configurar o perfil da função usando um PowerShell cmdlet. Para saber como, consulte a documentação do AWS Tools for Windows PowerShell.

Depois de configurar o perfil da função, faça referência ao perfil em seu aplicativo. Há várias maneiras de fazer isso, uma delas é usando AWSConfigso. AWSProfileNomeie a propriedade da mesma forma que você faria com outros perfis de credenciais.

A AWS Security Token Servicemontagem (AWSSDK. SecurityToken) fornece o suporte SAML para obter AWS credenciais. Para usar as credenciais da conta de usuário federado, certifique-se de que esse assembly esteja disponível para seu aplicativo.

Especificar funções ou credenciais temporárias

Para aplicativos executados em EC2 instâncias da Amazon, a maneira mais segura de gerenciar credenciais é usar funções do IAM, conforme descrito emConceder acesso utilizando um perfil do IAM.

Para cenários do aplicativo em que o executável do software está disponível para usuários de fora da organização, recomendamos que você projete o software para usar credenciais de segurança temporárias. Além de fornecer acesso restrito aos AWS recursos, essas credenciais têm a vantagem de expirar após um período de tempo especificado. Para obter mais informações sobre as credenciais de segurança temporárias, consulte:

- Credenciais de segurança temporárias
- Banco de identidades do Amazon Cognito

Usar credenciais de proxy

Se seu software se comunicar AWS por meio de um proxy, você poderá especificar as credenciais para o proxy usando a `ProxyCredentials` propriedade da `Config` classe de um serviço. A classe `Config` de um serviço normalmente faz parte do namespace primário do serviço. Os exemplos incluem o seguinte: AmazonCloudDirectoryConfigna Amazon. CloudDirectorynamespace e AmazonGameLiftConfigna Amazon. GameLiftnamespace.

Para o Amazon S3, por exemplo, você pode usar um código semelhante ao seguinte, onde `SecurelyStoredUserName` e `SecurelyStoredPassword` são o nome de usuário e a senha do proxy especificados em um NetworkCredentialobjeto.

```
AmazonS3Config config = new AmazonS3Config();
config.ProxyCredentials = new NetworkCredential(SecurelyStoredUserName,
 SecurelyStoredPassword);
```

> **Note**
>
> Versões anteriores do SDK utilizavam `ProxyUsername` e `ProxyPassword`, mas estas propriedades estão obsoletas.

Novas tentativas e tempos limite

O AWS SDK para .NET permite que você configure o número de novas tentativas e os valores de tempo limite para solicitações HTTP aos AWS serviços. Se os valores padrão para novas tentativas e tempos limite não forem apropriados, ajuste-os conforme os seus requisitos específicos, mas é importante entender como essa ação afetará o comportamento do aplicativo.

Para determinar quais valores usar para tentativas e tempos limite, considere o seguinte:

- Como o aplicativo AWS SDK para .NET e seu aplicativo devem responder quando a conectividade de rede se degrada ou um AWS serviço está inacessível? Deseja que a chamada falhe rapidamente, ou é apropriado que a chamada continue tentando em seu nome?
- O seu aplicativo é um aplicativo ou site voltado para o usuário e que deve ser responsivo, ou ele é um trabalho de processamento em segundo plano com maior tolerância a latências?
- O aplicativo é implantado em uma rede confiável com baixa latência ou é implantado em um local remoto com conectividade incerta?

Novas tentativas

Visão geral do

Eles AWS SDK para .NET podem repetir solicitações que falham devido à limitação do servidor ou à queda de conexões. Há duas propriedades das classes de configuração de serviço que você pode usar para especificar o comportamento de novas tentativas de um cliente de serviço. As classes de configuração de serviço herdam essas propriedades do abstrato Amazon.Runtime. ClientConfigclasse da referência da AWS SDK para .NET API:

- `RetryMode`especifica um dos três modos de repetição, definidos no Amazon.Runtime. RequestRetryModeenumeração.

  O valor padrão do seu aplicativo pode ser controlado usando a variável de `AWS_RETRY_MODE` ambiente ou a configuração retry_mode no arquivo de configuração compartilhado AWS .

- `MaxErrorRetry` especifica o número de novas tentativas permitidas no nível do cliente de serviço; o SDK executa novas tentativas da operação pelo número especificado de vezes antes de causar uma falha e gerar uma exceção.

  O valor padrão do seu aplicativo pode ser controlado usando a variável de `AWS_MAX_ATTEMPTS` ambiente ou a configuração max_attempts no arquivo de AWS configuração compartilhado.

As descrições detalhadas dessas propriedades podem ser encontradas no resumo Amazon.Runtime. ClientConfigclasse da Referência da AWS SDK para .NET API. Cada valor de `RetryMode` corresponde, por padrão, a um valor específico de `MaxErrorRetry`, conforme mostrado na tabela a seguir.

RetryMode	Correspondente MaxErrorR etry (Amazon DynamoDB)	Correspondente MaxErrorR etry (todos os outros)
Legada	10	4
Standard	10	2
Adaptativo (experimental)	10	2

Comportamento

Quando seu aplicativo é iniciado

Quando seu aplicativo é iniciado, os valores padrão para `RetryMode` e `MaxErrorRetry` são configurados pelo SDK. Esses valores padrão são usados quando você cria um cliente de serviço, a menos que você especifique outros valores.

- Se as propriedades não estiverem definidas em seu ambiente, o padrão para `RetryMode` será configurado como Legacy e o padrão para `MaxErrorRetry` será configurado com o valor correspondente da tabela anterior.

- Se o modo de novas tentativas tiver sido definido em seu ambiente, esse valor será usado como padrão para `RetryMode`. O padrão para `MaxErrorRetry` é configurado com o valor correspondente da tabela anterior, a menos que o valor para o máximo de erros também tenha sido definido em seu ambiente (descrito a seguir).
- Se o valor para o máximo de erros tiver sido definido em seu ambiente, esse valor será usado como padrão para `MaxErrorRetry`. O Amazon DynamoDB é a exceção a essa regra; o valor padrão do DynamoDB para `MaxErrorRetry` é sempre o valor da tabela anterior.

À medida que seu aplicativo é executado

Ao criar um cliente de serviço, você pode usar os valores padrão para `RetryMode` e `MaxErrorRetry`, conforme descrito anteriormente, ou pode especificar outros valores. Para especificar outros valores, crie e inclua um objeto de configuração de serviço, como AmazonDynamoDBConfigou Amazon, SQSConfig ao criar o cliente de serviço.

Não é possível alterar os valores de um cliente de serviço depois que ele tiver sido criado.

Considerações

Quando ocorre uma nova tentativa, a latência da solicitação é aumentada. Configure as tentativas com base nos limites de latência total da solicitação e taxas de erros do aplicativo.

Tempos limite

AWS SDK para .NET Isso permite que você configure os tempos limite de solicitação no nível do cliente de serviço e por chamada de método. Há dois mecanismos para configurar tempos limite, que serão abordados nas seções subsequentes:

- Se você estiver usando chamadas assíncronas, poderá utilizar o parâmetro `CancellationToken` do método.
- Se você estiver usando chamadas síncronas no.NET Framework, poderá usar as `ReadWriteTimeout` propriedades `Timeout` e do abstrato Amazon.Runtime. ClientConfigclasse.

Usar o parâmetro **`CancellationToken`** para tempos limite

O AWS SDK para .NET permite que você configure tempos limite de solicitação em chamadas assíncronas usando o parâmetro. `CancellationToken` O trecho de código a seguir mostra

um exemplo. O código lançará `System.Threading.Tasks.TaskCanceledException` se a solicitação não for concluída em 10 segundos.

```
string bucketName = "amzn-s3-demo-bucket";
string path = "pathToBucket";
using (var amazonS3Client = new AmazonS3Client(new AmazonS3Config()))
{
    // Cancel request after 10 seconds
    CancellationTokenSource cancellationTokenSource = new
 CancellationTokenSource(TimeSpan.FromMilliseconds(10000));
    CancellationToken cancellationToken = cancellationTokenSource.Token;
    ListObjectsV2Request listRequestV2 = new()
    {
        BucketName = bucketName,
        Prefix = path,
    };

    ListObjectsV2Response listResponseV2 = await
 amazonS3Client.ListObjectsV2Async(listRequestV2, cancellationToken);
}
```

Usar as propriedades **`Timeout`** e **`ReadWriteTimeout`**para tempos limite

> **Note**
>
> A propriedade `Timeout` não afeta as chamadas assíncronas. Se você estiver usando chamadas assíncronas, consulte Usar o parâmetro `CancellationToken` para tempos limite em vez disso.

O AWS SDK para .NET permite que você configure os valores de tempo limite da solicitação e do tempo read/write limite do soquete no nível do cliente de serviço. Esses valores são especificados nas `Timeout` `ReadWriteTimeout` propriedades abstratas do Amazon.Runtime. ClientConfigclasse. Esses valores são transmitidos como `ReadWriteTimeout` propriedades `Timeout` e dos HttpWebRequestobjetos criados pelo objeto cliente do AWS serviço. Por padrão, o valor `Timeout` é 100 segundos e o valor `ReadWriteTimeout` é 300 segundos.

Quando a sua rede estiver com latência alta, ou existirem condições que gerem uma nova tentativa para uma operação, utilizar valores de tempo limite longos e um alto número de tentativas pode fazer com que algumas operações do SDK pareçam indiferentes.

> **Note**
>
> A versão do AWS SDK para .NET que tem como alvo a biblioteca de classes portátil (PCL) usa a HttpClientclasse em vez da `HttpWebRequest` classe e oferece suporte somente à propriedade Timeout.

Veja a seguir as exceções aos valores de tempo limite padrão. Estes valores são substituídos ao definir explicitamente os valores de tempo limite.

- `Timeout`e `ReadWriteTimeout` são definidos com os valores máximos se o método que está sendo chamado fizer upload de um stream, como AmazonS3Client. PutObjectAsync(), cliente Amazon S3. UploadPartAsync(), AmazonGlacierClient. UploadArchiveAsync() e assim por diante.
- As versões do AWS SDK para .NET conjunto do.NET Framework têm como alvo os valores máximos `ReadWriteTimeout` para todos os objetos `Timeout` e clientes do AmazonS3. AmazonGlacierClient
- As versões do AWS SDK para .NET que têm como alvo a biblioteca de classes portátil (PCL) e o.NET Core são definidas como o valor máximo `Timeout` para todos os objetos e clientes do AmazonS3. AmazonGlacierClient

O exemplo a seguir mostra como especificar o modo de repetição padrão, no máximo 3 tentativas, um tempo limite de 10 segundos e um read/write tempo limite de 10 segundos (se aplicável). O construtor AmazonS3Client recebe um objeto AmazonS3Config.

```
var s3Client = new AmazonS3Client(
  new AmazonS3Config
  {
    Timeout = TimeSpan.FromSeconds(10),
    // NOTE: The following property is obsolete for
    //       versions of the AWS SDK para .NET that target .NET Core.
    ReadWriteTimeout = TimeSpan.FromSeconds(10),
    RetryMode = RequestRetryMode.Standard,
    MaxErrorRetry = 3
  });
```

Observabilidade

Observabilidade é a medida em que o estado atual de um sistema pode ser identificado com base nos dados que ele emite. Os dados emitidos são chamados, com frequência, de telemetria.

Eles AWS SDK para .NET podem fornecer dois sinais, métricas e rastreamentos de telemetria comuns, além de registros. Você pode conectar um TelemetryProviderpara enviar dados de telemetria para um back-end de observabilidade (como a AWS X-RayAmazon CloudWatch) e, em seguida, agir de acordo com eles.

Por padrão, os sinais de telemetria permanecem desabilitados no SDK. Este tópico explica como habilitar e configurar a saída de telemetria.

Recursos adicionais do

Para acessar mais informações sobre como habilitar e usar a observabilidade, consulte os seguintes recursos:

- OpenTelemetry
- A postagem do blog Enhancing Observability in the AWS SDK para .NET with. OpenTelemetry
- A postagem do blog anunciando a disponibilidade geral das AWS OpenTelemetry bibliotecas.NET.
- Exportadores para OpenTelemetry
- Para exemplos de observabilidade no Ferramentas da AWS para PowerShell, consulte Observabilidade no Guia de Ferramentas do PowerShell Usuário.

Configurar um `TelemetryProvider`

Você pode configurar um `TelemetryProvider` em sua aplicação globalmente para todos os clientes de serviço ou para clientes individuais, conforme mostrado nos exemplos a seguir. A seção the section called “Provedores de telemetria” contém informações sobre implementações de telemetria, incluindo informações sobre implementações fornecidas com o SDK.

Configurar o provedor de telemetria global padrão

Por padrão, todo cliente de serviço tenta usar o provedor de telemetria disponível globalmente. Dessa forma, é possível definir o provedor uma vez e todos os clientes o usarão. Isso deve ser feito apenas uma vez, antes de criar qualquer cliente de serviço.

O trecho de código a seguir mostra como definir o provedor de telemetria global. Depois, ele cria um cliente de serviço Amazon S3 e tenta realizar uma operação que falha. O código adiciona rastreamento e métricas à aplicação. Esse código usa os seguintes NuGet pacotes: `OpenTelemetry.Exporter.Console` `OpenTelemetry.Instrumentation.AWS` e.

> **Note**
>
> Se você estiver usando Centro de Identidade do AWS IAM para autenticação, não se esqueça de também adicionar `AWSSDK.SSO` `AWSSDK.SSOOIDC` e.

```
using Amazon.S3;
using OpenTelemetry;
using OpenTelemetry.Metrics;
using OpenTelemetry.Resources;
using OpenTelemetry.Trace;

Sdk.CreateTracerProviderBuilder()
    .ConfigureResource(e => e.AddService("DemoOtel"))
    .AddAWSInstrumentation()
    .AddConsoleExporter()
    .Build();

Sdk.CreateMeterProviderBuilder()
    .ConfigureResource(e => e.AddService("DemoOtel"))
    .AddAWSInstrumentation()
    .AddConsoleExporter()
    .Build();

var s3Client = new AmazonS3Client();

try
{
    var listBucketsResponse = await s3Client.ListBucketsAsync();
    // Attempt to delete a bucket that doesn't exist.
    var deleteBucketResponse = await s3Client.DeleteBucketAsync("amzn-s3-demo-bucket");
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}
```

```
Console.Read();
```

Configurar um provedor de telemetria para um cliente de serviço específico

É possível configurar um cliente de serviço individual com um provedor de telemetria específico (diferente do global). Para isso, use a classe `TelemetryProvider` do objeto Config de um construtor de cliente de serviço. Por exemplo, consulte AmazonS3Config e procure a propriedade `TelemetryProvider`. Consulte the section called “Provedores de telemetria” para acessar informações sobre implementações personalizadas de telemetria.

Tópicos

- Métricas
- Provedores de telemetria

Métricas

A tabela a seguir lista as métricas de telemetria que o SDK emite. Configure um provedor de telemetria para tornar as métricas observáveis.

Quais métricas são emitidas?

Nome da métrica	Unidades	Tipo	Atributos	Descrição
client.call.duration	s	Histograma	rpc.service, rpc.method	Duração geral da chamada (incluindo novas tentativas e tempo para enviar ou receber a solicitação e o corpo da resposta)
client.uptime	s	Histograma	rpc.service	A quantidade de tempo desde que um cliente foi criado
client.call.attempts	{tentativa}	MonotonCounter	rpc.service, rpc.method	O número de tentativas para uma operação individual
client.call.errors	{erro}	MonotonCounter	rpc.service, rpc.method, exceção.type	O número de erros de uma operação

Nome da métrica	Unidades	Tipo	Atributos	Descrição
client.call.attempt duration	s	Histogra a	rpc.service, rpc.method	O tempo necessário para se conectar ao serviço, enviar a solicitação e recuperar o código de status e os cabeçalhos HTTP (incluindo o tempo de espera para envio)
client.call.resolve_endpoint_duration	s	Histogra a	rpc.service, rpc.method	O tempo necessário para resolver um endpoint (resolvedor de endpoint, não DNS) para a solicitação
client.call.serialization_duration	s	Histogra a	rpc.service, rpc.method	O tempo necessário para serializar o corpo da mensagem
client.call.deserialization_duration	s	Histogra a	rpc.service, rpc.method	O tempo necessário para desserializar o corpo da mensagem
client.call.auth.signing_duration	s	Histogra a	rpc.service, rpc.method	O tempo necessário para assinar uma solicitação
client.call.auth.resolve_identity_duration	s	Histogra a	rpc.service, rpc.method	O tempo necessário para adquirir uma identidade (como AWS credenciais ou um token de portador) de um provedor de identidade
client.http.bytes_sent	Por	Monoton Counter	endereço do servidor	O número total de bytes enviados pelo cliente HTTP
client.http.bytes_recebido	Por	Monoton Counter	endereço do servidor	O número total de bytes recebidos pelo cliente HTTP

A seguir estão as descrições das colunas:

- Nome da métrica — O nome da métrica emitida.
- Unidades — A unidade de medida da métrica. As unidades são fornecidas na notação UCUM com distinção entre maiúsculas e minúsculas (“c/s”).

- Tipo — O tipo de instrumento usado para capturar a métrica.
- Atributos — O conjunto de atributos (dimensões) emitidos com a métrica.
- Descrição — Uma descrição do que a métrica está medindo.

Provedores de telemetria

O SDK fornece uma implementação de OpenTelemetrycomo provedor de telemetria, descrita na próxima seção.

Se você tem requisitos específicos de telemetria, já tem uma solução de telemetria em mente ou precisa de um controle refinado sobre como os dados de telemetria são capturados e processados, você também pode implementar seu próprio provedor de telemetria.

Registre sua própria implementação com a TelemetryProviderturma. A seguir está um exemplo simples de como registrar seu próprio `TracerProvider` `MeterProvider` e.

```
using Amazon;
using Amazon.Runtime.Telemetry;
using Amazon.Runtime.Telemetry.Metrics;
using Amazon.Runtime.Telemetry.Tracing;

public class CustomTracerProvider : TracerProvider
{
    // Implement custom tracing logic here
}
public class CustomMeterProvider : MeterProvider
{
    // Implement custom metrics logic here
}

// Register custom implementations
AWSConfigs.TelemetryProvider.RegisterTracerProvider(new CustomTracerProvider());
AWSConfigs.TelemetryProvider.RegisterMeterProvider(new CustomMeterProvider());
```

Tópicos

- Configurar o provedor OpenTelemetry de telemetria baseado

Configurar o provedor OpenTelemetry de telemetria baseado

AWS SDK para .NET Isso inclui a implementação de um provedor de telemetria OpenTelemetry baseado. Para acessar detalhes sobre como definir esse provedor como o provedor global de telemetria, consulte Configurar um `TelemetryProvider`. Para usar esse provedor de telemetria, você precisa dos seguintes recursos em seu projeto:

- O pacote OpenTelemetry NuGet .Instrumentation.aws.
- Um exportador de telemetria, como OTLP ou Console. Para obter mais informações, consulte Exportadores na OpenTelemetry documentação.

A OpenTelemetry implementação incluída no SDK pode ser configurada para reduzir a quantidade de rastreamento de solicitações, credenciais e compactação HTTPS. Para fazer isso, defina a opção `SuppressDownstreamInstrumentation` como `true`, semelhante à seguinte:

```
Sdk.CreateTracerProviderBuilder()
    .ConfigureResource(e => e.AddService("DemoOtel"))
    .AddAWSInstrumentation(options => options.SuppressDownstreamInstrumentation = true)
    .AddConsoleExporter()
    .Build();
```

Para obter informações adicionais sobre esse provedor, consulte a postagem do blog Enhancing Observability in the AWS SDK para .NET with. OpenTelemetry

Informações adicionais sobre usuários e perfis

Para fazer o desenvolvimento do.NET AWS ou executar aplicativos.NET em AWS, você precisa ter uma combinação de usuários, conjuntos de permissões e funções de serviço que sejam apropriadas para essas tarefas.

Os usuários específicos, os conjuntos de permissões e os perfis de serviço que você cria e a maneira como você os utiliza dependerão dos seus dos seus aplicativos. Veja a seguir algumas informações adicionais sobre por que eles podem ser usados e como criá-los.

Usuários e conjuntos de permissões

Embora seja possível usar uma conta de usuário do IAM com credenciais de longo prazo para acessar os serviços da AWS , essa não é mais uma prática recomendada e deve ser evitada. Mesmo durante o desenvolvimento, é uma prática recomendada criar usuários e conjuntos de permissões

Centro de Identidade do AWS IAM e usar credenciais temporárias fornecidas por uma fonte de identidade.

Para desenvolvimento, você pode usar o usuário que criou ou recebeu em Autenticando com AWS. Se você tiver Console de gerenciamento da AWS as permissões apropriadas, também poderá criar conjuntos de permissões diferentes com menos privilégios para esse usuário ou criar novos usuários especificamente para projetos de desenvolvimento, fornecendo conjuntos de permissões com menos privilégios. A ação que você escolher, se for o caso, dependerá das circunstâncias.

Para obter mais informações sobre esses usuários e conjuntos de permissões e como criá-los, consulte Autenticação AWS SDKs e acesso no Guia de referência de ferramentas e Introdução no Guia do Centro de Identidade do AWS IAM usuário.

Perfis de serviço

Você pode configurar uma função AWS de serviço para acessar AWS serviços em nome dos usuários. Esse tipo de acesso é apropriado se várias pessoas estiverem executando seu aplicativo remotamente; por exemplo, em uma EC2 instância da Amazon que você criou para essa finalidade.

O processo de criação de um perfil de serviço varia de acordo com a situação, mas é basicamente o seguinte.

1. Faça login no Console de gerenciamento da AWS e abra o console do IAM em https://console.aws.amazon.com/iam/.
2. Selecione Funções e, depois, Criar função.
3. Escolha o AWS serviço, localize e selecione EC2(por exemplo) e, em seguida, escolha o caso de EC2uso (por exemplo).
4. Escolha Avançar: Permissões e selecione as políticas apropriadas para os AWS serviços que seu aplicativo usará.

   > **Warning**
   >
   > NÃO escolha a AdministratorAccesspolítica porque ela permite permissões de leitura e gravação em quase tudo na sua conta.

5. Escolha Próximo: Tags e insira as tags que você desejar.

   Você pode encontrar informações sobre tags em Controlar o acesso usando tags AWS de recursos no Guia do usuário do IAM.

6. Escolha Próximo: Revisar para fornecer um nome de perfil e uma descrição de perfil. Então, escolha Criar perfil.

Você pode encontrar informações de alto nível sobre os perfis do IAM em Identidades (usuários, grupos e perfis) no Guia do usuário do IAM. Encontre informações detalhadas sobre perfis nesse tópico Perfis do IAM.

Informações adicionais sobre perfis

- Use perfis do IAM para tarefas do Amazon Elastic Container Service (Amazon ECS).
- Use funções do IAM para aplicativos que estão sendo executados em EC2 instâncias da Amazon.

Configuração avançada para seu AWS SDK para .NET projeto

Os tópicos desta seção contêm informações sobre tarefas e métodos de configurações adicionais que podem ser do seu interesse.

Tópicos

- Usando AWSSDK .Extensions. NETCore.Configuração e interface IConfiguration
- Configuração de outros parâmetros do aplicativo
- Referência de arquivos de configuração para AWS SDK para .NET

Usando AWSSDK .Extensions. NETCore.Configuração e interface IConfiguration

(Este tópico era anteriormente intitulado “Configurando o AWS SDK para .NET com o.NET Core”)

Uma das principais alterações no .NET Core é a retirada do `ConfigurationManager` e dos arquivos padrão `app.config` e `web.config` usados nos aplicativos .NET Framework e ASP.NET.

A configuração no .NET Core é baseada em pares de chave/valor estabelecidos pelos provedores de configuração. Os provedores de configuração leem dados de configuração em pares de chave/valor de uma variedade de fontes de configuração, incluindo argumentos de linha de comando, arquivos de diretório, variáveis de ambiente e arquivos de configurações.

> **Note**
>
> Para obter mais informações, consulte Configuração no ASP.NET Core.

Para facilitar o uso do AWS SDK para .NET com o.NET Core, você pode usar as AWSSDK.Extensions. NETCore NuGet .Pacote de configuração. Como muitas bibliotecas.NET Core, ela adiciona métodos de extensão à `IConfiguration` interface para facilitar a obtenção da AWS configuração.

O código-fonte desse pacote está GitHub emhttps://github.com/aws/aws-sdk-net/tree/main/extensions/src/AWSSDK.Extensions.NETCore.Setup.

Usando AWSSDK .Extensions. NETCore.Configuração

Suponha que você crie um aplicativo ASP.NET Core Model-View-Controller (MVC), o que pode ser feito com o modelo de aplicativo Web ASP.NET Core no Visual Studio ou executando na `dotnet new mvc ...` CLI do.NET Core. Quando você cria tal aplicativo, o construtor do `Startup.cs` lida com a configuração lendo várias fontes de entrada de provedores de configuração, como o `appsettings.json`.

```
public Startup(IConfiguration configuration)
{
    Configuration = configuration;
}
```

Para usar o `Configuration` objeto para obter as AWSopções, primeiro adicione o `AWSSDK.Extensions.NETCore.Setup` NuGet pacote. Em seguida, adicione as opções ao arquivo de configuração, conforme descrito a seguir.

Observe que um dos arquivos adicionados ao seu projeto é `appsettings.Development.json`. Isso corresponde a um `EnvironmentName` definido como Desenvolvimento. Durante o desenvolvimento, você coloca sua configuração nesse arquivo, que só é lido durante o teste local. Quando você implanta uma instância do Amazon EC2 que foi `EnvironmentName` configurada como Produção, esse arquivo é ignorado e volta para as AWS SDK para .NET credenciais e a região do IAM que estão configuradas para a instância do Amazon EC2.

As definições de configuração a seguir mostram exemplos dos valores que você pode incluir no arquivo `appsettings.Development.json` em seu projeto para fornecer as configurações da AWS .

```
{
  "AWS": {
    "Profile": "local-test-profile",
    "Region": "us-west-2"
  },
  "SupportEmail": "TechSupport@example.com"
}
```

Para acessar uma configuração em um arquivo CSHTML, use a diretiva `Configuration`.

```
@using Microsoft.Extensions.Configuration
@inject IConfiguration Configuration

<h1>Contact</h1>

<p>
    <strong>Support:</strong> <a
 href='mailto:@Configuration["SupportEmail"]'>@Configuration["SupportEmail"]</a><br />
</p>
```

Para acessar as AWS opções definidas no arquivo a partir do código, chame o método de `GetAWSOptions` extensão adicionado `IConfiguration` a.

Para construir um cliente de serviço a partir dessas opções, chame `CreateServiceClient`. O código de exemplo a seguir mostra como criar um cliente de serviço do Amazon S3. (Certifique-se de adicionar o AWSSDK NuGet pacote.S3 ao seu projeto.)

```
var options = Configuration.GetAWSOptions();
IAmazonS3 client = options.CreateServiceClient<IAmazonS3>();
```

Também é possível criar vários clientes de serviço com configurações incompatíveis ao usar várias entradas no arquivo `appsettings.Development.json`, conforme mostrado nos exemplos a seguir, em que a configuração de `service1` inclui a região `us-west-2` e a configuração de `service2` inclui o URL do endpoint especial.

```
{
  "service1": {
```

```
    "Profile": "default",
    "Region": "us-west-2"
  },
  "service2": {
    "Profile": "default",
    "ServiceURL": "URL"
  }
}
```

Depois, você pode obter as opções para um serviço específico usando a entrada no arquivo JSON. Por exemplo, para obter as configurações para `service1`, use o seguinte.

```
var options = Configuration.GetAWSOptions("service1");
```

Valores permitidos no arquivo appsettings

Os valores de configuração do aplicativo a seguir podem ser definidos no arquivo `appsettings.Development.json`. Os nomes de campo devem usar a capitalização mostrada. Para obter mais detalhes sobre essas configurações, consulte a classe `AWS.Runtime.ClientConfig`.

- Região
- Perfil
- ProfilesLocation
- SignatureVersion
- RegionEndpoint
- UseHttp
- ServiceURL
- AuthenticationRegion
- AuthenticationServiceName
- MaxErrorRetry
- LogResponse
- BufferSize
- ProgressUpdateInterval
- ResignRetries
- AllowAutoRedirect

- LogMetrics
- DisableLogging
- UseDualstackEndpoint

Injeção de dependência no núcleo do ASP.NET

As AWSSDKextensões. NETCoreO NuGet pacote.Setup também se integra a um novo sistema de injeção de dependência no ASP.NET Core. O método `ConfigureServices` na classe `Startup` do seu aplicativo é o local onde os serviços MVC são adicionados. Se o aplicativo usa a Estrutura de entidade, também é aqui que ela é inicializada.

```
public void ConfigureServices(IServiceCollection services)
{
    // Add framework services.
    services.AddMvc();
}
```

> **Note**
>
> O histórico sobre a injeção de dependência no .NET Core está disponível no site de documentação do .NET Core.

O `AWSSDK.Extensions.NETCore.Setup` NuGet pacote adiciona novos métodos de extensão `IServiceCollection` que você pode usar para adicionar AWS serviços à injeção de dependência. O código a seguir mostra como adicionar as AWS opções que são lidas `IConfiguration` para adicionar o Amazon S3 e o DynamoDB à lista de serviços. (Certifique-se de adicionar os DBv2 NuGet pacotes AWSSDK.S3 e AWSSDK.Dynamo ao seu projeto.)

```
public void ConfigureServices(IServiceCollection services)
{
    // Add framework services.
    services.AddMvc();

    services.AddDefaultAWSOptions(Configuration.GetAWSOptions());
    services.AddAWSService<IAmazonS3>();
    services.AddAWSService<IAmazonDynamoDB>();
}
```

Agora, se os controladores MVC usam `IAmazonS3` ou `IAmazonDynamoDB` como parâmetros em seus construtores, o sistema de injeção de dependência passa nesses serviços.

```
public class HomeController : Controller
{
    IAmazonS3 S3Client { get; set; }

    public HomeController(IAmazonS3 s3Client)
    {
        this.S3Client = s3Client;
    }

    ...

}
```

Configuração de outros parâmetros do aplicativo

> **Note**
>
> As informações neste tópico são específicas para projetos baseados no .NET Framework. Os arquivos `App.config` e `Web.config` não estão presentes por padrão em projetos baseados no .NET Core.

Abra para ver o conteúdo do .NET Framework

Há diversos parâmetros do aplicativo que você pode configurar:

- `AWSLogging`
- `AWSLogMetrics`
- `AWSRegion`
- `AWSResponseLogging`
- `AWS.DynamoDBContext.TableNamePrefix`
- `AWS.S3.UseSignatureVersion4`
- `AWSEndpointDefinition`
- AWS Endpoints gerados por serviços

Esses parâmetros podem ser configurados no arquivo `App.config` ou `Web.config` do aplicativo. Embora você também possa configurá-los com a AWS SDK para .NET API, recomendamos que você use o `.config` arquivo do aplicativo. Ambas as abordagens são descritas aqui.

Para obter mais informações sobre o uso do elemento `<aws>` conforme descrito posteriormente neste tópico, consulte Referência de arquivos de configuração para o AWS SDK para .NET.

AWSLogging

Configura a forma como o SDK registra os eventos, caso registre. Por exemplo, a abordagem recomendada é usar o elemento `<logging>`, que é um elemento filho do elemento `<aws>`:

```
<aws>
  <logging logTo="Log4Net"/>
</aws>
```

Alternativa:

```
<add key="AWSLogging" value="log4net"/>
```

Os valores possíveis são:

`None`

Desative o registro de eventos do . Esse é o padrão.

`log4net`

Registre usando o log4net.

`SystemDiagnostics`

Registre usando a classe `System.Diagnostics`.

Defina diversos valores para o atributo `logTo`, separado por vírgulas. O exemplo a seguir define ambos os registros `log4net` e `System.Diagnostics` no arquivo `.config`:

```
<logging logTo="Log4Net, SystemDiagnostics"/>
```

Alternativa:

```
<add key="AWSLogging" value="log4net, SystemDiagnostics"/>
```

Como alternativa, usando a AWS SDK para .NET API, combine os valores da LoggingOptionsenumeração e defina a AWSConfigs propriedade .Logging:

```
AWSConfigs.Logging = LoggingOptions.Log4Net | LoggingOptions.SystemDiagnostics;
```

As alterações nessa configuração entrarão em vigor somente para novas instâncias AWS do cliente.

AWSLogMétricas

Especifica se o SDK deve ou não registrar as métricas de desempenho. Para definir a configuração do registro das métricas no arquivo `.config`, defina o valor do atributo `logMetrics` no elemento `<logging>`, que é um elemento filho do elemento `<aws>`:

```
<aws>
  <logging logMetrics="true"/>
</aws>
```

Como alternativa, defina a chave `AWSLogMetrics` na seção`<appSettings>`:

```
<add key="AWSLogMetrics" value="true">
```

Como alternativa, para definir o registro de métricas com a AWS SDK para .NET API, defina AWSConfigso. LogMetricspropriedade:

```
AWSConfigs.LogMetrics = true;
```

Essa definição configura propriedade padrão `LogMetrics` para todos os clientes/configurações. As alterações nessa configuração entrarão em vigor somente para novas instâncias AWS do cliente.

AWSRegion

Configura a AWS região padrão para clientes que não especificaram explicitamente uma região. Para definir a região no arquivo `.config`, a abordagem recomendada é definir o valor do atributo `region` no elemento `aws`:

```
<aws region="us-west-2"/>
```

Como alternativa, defina a chave AWSRegion na seção`<appSettings>`:

```
<add key="AWSRegion" value="us-west-2"/>
```

Como alternativa, para definir a região com a AWS SDK para .NET API, defina AWSConfigso. AWSRegionpropriedade:

```
AWSConfigs.AWSRegion = "us-west-2";
```

Para obter mais informações sobre como criar um AWS cliente para uma região específica, consulte Seleção de AWS região. As alterações nessa configuração entrarão em vigor somente para novas instâncias AWS do cliente.

AWSResponseRegistro

Configura quando o SDK deve registrar respostas de serviços. Os valores possíveis são:

`Never`

Nunca registrar as respostas de serviços. Esse é o padrão.

`Always`

Sempre registrar as respostas de serviços.

`OnError`

Somente registrar as respostas de serviços quando ocorrer um erro.

Para definir a configuração do registro de serviços no arquivo `.config`, a abordagem recomendada é definir o valor do atributo `logResponses` no elemento `<logging>`, que é um elemento filho do elemento `<aws>`:

```
<aws>
  <logging logResponses="OnError"/>
</aws>
```

Como alternativa, defina a chave de AWSResponseregistro na `<appSettings>` seção:

```
<add key="AWSResponseLogging" value="OnError"/>
```

Como alternativa, para definir o registro de serviços com a AWS SDK para .NET API, defina AWSConfigso. ResponseLoggingpropriedade de um dos valores da ResponseLoggingOptionenumeração:

```
AWSConfigs.ResponseLogging = ResponseLoggingOption.OnError;
```

As alterações desta configuração entram em vigor imediatamente.

`AWS.DynamoDBContext.TableNamePrefix`

Configura a `TableNamePrefix` padrão que o `DynamoDBContext` usará, se não for configurada manualmente.

Para definir o prefixo do nome da tabela no arquivo `.config`, a abordagem recomendada é definir o valor do atributo `tableNamePrefix` no elemento `<dynamoDBContext>`, que é um elemento filho do elemento `<dynamoDB>`, que por sua vez é um elemento filho do elemento `<aws>`:

```
<dynamoDBContext tableNamePrefix="Test-"/>
```

Como alternativa, defina a chave `AWS.DynamoDBContext.TableNamePrefix` na seção`<appSettings>`:

```
<add key="AWS.DynamoDBContext.TableNamePrefix" value="Test-"/>
```

Como alternativa, para definir o prefixo do nome da tabela com a AWS SDK para .NET API, defina a propriedade AWSConfigsDBContextTableNamePrefix.Dynamo:

```
AWSConfigs.DynamoDBContextTableNamePrefix = "Test-";
```

As alterações desta configuração serão implementadas somente em instâncias recentemente criadas do `DynamoDBContextConfig` e `DynamoDBContext`.

`AWS.S3.UseSignatureVersion4`

Configura se o cliente do Amazon S3 deve ou não usar a assinatura do Signature versão 4 com as solicitações.

Para definir a assinatura da versão 4 para o Amazon S3 no arquivo `.config`, a abordagem recomendada é definir o valor do atributo `useSignatureVersion4` do elemento `<s3>`, que é um elemento filho do elemento `<aws>`:

```
<aws>
  <s3 useSignatureVersion4="true"/>
</aws>
```

Como alternativa, defina a chave `AWS.S3.UseSignatureVersion4` como `true` na seção `<appSettings>`:

```
<add key="AWS.S3.UseSignatureVersion4" value="true"/>
```

Como alternativa, para definir a assinatura da versão 4 com a AWS SDK para .NET API, defina a propriedade AWSConfigs.S3 UseSignatureVersion 4 como: `true`

```
AWSConfigs.S3UseSignatureVersion4 = true;
```

Por padrão, esta configuração é `false`, mas o Signature versão 4 pode ser usado como padrão em alguns casos ou em algumas regiões. Quando a configuração é `true`, o Signature versão 4 será usada em todas as solicitações. As alterações desta configuração entram em vigor somente para novas instâncias de cliente do Amazon S3.

AWSEndpointDefinição

Configura se o SDK deve ou não usar um arquivo de configuração personalizado que define as regiões e os endpoints.

Para definir o arquivo de definição do endpoint no arquivo `.config`, recomendamos definir o valor do atributo `endpointDefinition` no elemento `<aws>`.

```
<aws endpointDefinition="c:\config\endpoints.json"/>
```

Como alternativa, você pode definir a chave de AWSEndpointdefinição na `<appSettings>` seção:

```
<add key="AWSEndpointDefinition" value="c:\config\endpoints.json"/>
```

Como alternativa, para definir o arquivo de definição do endpoint com a AWS SDK para .NET API, defina o. AWSConfigs EndpointDefinitionpropriedade:

```
AWSConfigs.EndpointDefinition = @"c:\config\endpoints.json";
```

Se nenhum nome de arquivo for fornecido, não será usado um arquivo de configuração personalizado. As alterações nessa configuração entrarão em vigor somente para novas instâncias AWS do cliente. O `endpoints.json` arquivo está disponível emhttps://github.com/aws/aws-sdk-net/blob/main/sdk/src/Core/endpoints.json.

AWS Endpoints gerados por serviços

Alguns AWS serviços geram seus próprios endpoints em vez de consumir um endpoint regional. Os clientes desses serviços consomem um URL de serviço específico ao serviço e aos recursos. Dois exemplos desses serviços são Amazon CloudSearch AWS IoT e. Os exemplos a seguir mostram como obter os endpoints para esses serviços.

Exemplo de CloudSearch endpoints da Amazon

O CloudSearch cliente da Amazon é usado para acessar o serviço de CloudSearch configuração da Amazon. Você usa o serviço CloudSearch de configuração da Amazon para criar, configurar e gerenciar domínios de pesquisa. Para criar um domínio de pesquisa, crie um CreateDomainRequestobjeto e forneça a `DomainName` propriedade. Crie um AmazonCloudSearchClientobjeto usando o objeto de solicitação. Chame o método CreateDomain. O CreateDomainResponseobjeto retornado da chamada contém uma `DomainStatus` propriedade que tem os `SearchService` pontos finais `DocService` e os pontos finais. Crie um AmazonCloudSearchDomainConfigobjeto e use-o para inicializar `DocService` as `SearchService` instâncias da AmazonCloudSearchDomainClientclasse.

```
// Create domain and retrieve DocService and SearchService endpoints
DomainStatus domainStatus;
using (var searchClient = new AmazonCloudSearchClient())
{
    var request = new CreateDomainRequest
    {
        DomainName = "testdomain"
    };
    domainStatus = searchClient.CreateDomain(request).DomainStatus;
    Console.WriteLine(domainStatus.DomainName + " created");
}

// Test the DocService endpoint
var docServiceConfig = new AmazonCloudSearchDomainConfig
{
    ServiceURL = "https://" + domainStatus.DocService.Endpoint
};
using (var domainDocService = new AmazonCloudSearchDomainClient(docServiceConfig))
{
    Console.WriteLine("Amazon CloudSearchDomain DocService client instantiated using
 the DocService endpoint");
    Console.WriteLine("DocService endpoint = " + domainStatus.DocService.Endpoint);
```

```
    using (var docStream = new FileStream(@"C:\doc_source\XMLFile4.xml",
 FileMode.Open))
    {
        var upload = new UploadDocumentsRequest
        {
            ContentType = ContentType.ApplicationXml,
            Documents = docStream
        };
        domainDocService.UploadDocuments(upload);
    }
}

// Test the SearchService endpoint
var searchServiceConfig = new AmazonCloudSearchDomainConfig
{
    ServiceURL = "https://" + domainStatus.SearchService.Endpoint
};
using (var domainSearchService = new
 AmazonCloudSearchDomainClient(searchServiceConfig))
{
    Console.WriteLine("Amazon CloudSearchDomain SearchService client instantiated using
 the SearchService endpoint");
    Console.WriteLine("SearchService endpoint = " +
 domainStatus.SearchService.Endpoint);

    var searchReq = new SearchRequest
    {
        Query = "Gambardella",
        Sort = "_score desc",
        QueryParser = QueryParser.Simple
    };
    var searchResp = domainSearchService.Search(searchReq);
}
```

AWS IoT Exemplo de endpoints

Para obter o endpoint para AWS IoT, crie um AmazonIoTClientobjeto e chame o DescribeEndPointmétodo. O DescribeEndPointResponseobjeto retornado contém `EndpointAddress` o. Crie um AmazonIotDataConfigobjeto, defina a `ServiceURL` propriedade e use o objeto para instanciar a AmazonIotDataClientclasse.

```
string iotEndpointAddress;
using (var iotClient = new AmazonIoTClient())
```

```
{
    var endPointResponse = iotClient.DescribeEndpoint();
    iotEndpointAddress = endPointResponse.EndpointAddress;
}

var ioTdocServiceConfig = new AmazonIotDataConfig
{
    ServiceURL = "https://" + iotEndpointAddress
};
using (var dataClient = new AmazonIotDataClient(ioTdocServiceConfig))
{
    Console.WriteLine("AWS IoTData client instantiated using the endpoint from the
 IotClient");
}
```

Referência de arquivos de configuração para AWS SDK para .NET

> **Note**
>
> As informações neste tópico são específicas para projetos baseados no .NET Framework. Os arquivos `App.config` e `Web.config` não estão presentes por padrão em projetos baseados no .NET Core.

Abra para ver o conteúdo do .NET Framework

Você pode usar um projeto `App.config` ou `Web.config` arquivo do.NET para especificar AWS configurações, como AWS credenciais, opções de registro, endpoints de AWS serviço e AWS regiões, bem como algumas configurações de AWS serviços, como Amazon DynamoDB, EC2 Amazon e Amazon S3. As informações a seguir descrevem como formatar de forma apropriada um arquivo `App.config` ou `Web.config` para especificar esses tipos de configurações.

> **Note**
>
> Embora você possa continuar usando o `<appSettings>` elemento em um `Web.config` arquivo `App.config` ou para especificar AWS configurações, recomendamos que você use os `<aws>` elementos `<configSections>` e conforme descrito posteriormente neste tópico. Para obter mais informações sobre o `<appSettings>` elemento, consulte os exemplos de `<appSettings>` elementos em Configurando seu AWS SDK para .NET aplicativo.

> **Note**
>
> Embora você possa continuar usando as seguintes propriedades de `AWSConfigs`classe em um arquivo de código para especificar AWS configurações, as propriedades a seguir estão obsoletas e podem não ser suportadas em versões futuras:
>
> - `DynamoDBContextTableNamePrefix`
> - `EC2UseSignatureVersion4`
> - `LoggingOptions`
> - `LogMetrics`
> - `ResponseLoggingOption`
> - `S3UseSignatureVersion4`
>
> Em geral, recomendamos que, em vez de usar propriedades de `AWSConfigs` classe em um arquivo de código para especificar AWS configurações, você use os `<aws>` elementos `<configSections>` e em um `Web.config` arquivo `App.config` ou para especificar AWS configurações, conforme descrito posteriormente neste tópico. Para obter mais informações sobre as propriedades anteriores, consulte os exemplos de `AWSConfigs` código em Configurando seu AWS SDK para .NET aplicativo.

Tópicos

- Declarando uma seção de AWS configurações
- Elementos permitidos
- Referência dos elementos

Declarando uma seção de AWS configurações

Você especifica AWS as configurações em um `Web.config` arquivo `App.config` ou de dentro do `<aws>` elemento. Antes de começar a usar o elemento `<aws>`, é necessário criar um elemento `<section>` (que é um elemento filho do elemento `<configSections>`) e definir o seu atributo `name` como `aws` e o atributo `type` como `Amazon.AWSSection, AWSSDK.Core`, conforme mostrado no seguinte exemplo:

```
<?xml version="1.0"?>
```

```
<configuration>
  ...
  <configSections>
    <section name="aws" type="Amazon.AWSSection, AWSSDK.Core"/>
  </configSections>
  <aws>
    <!-- Add your desired AWS settings declarations here. -->
  </aws>
  ...
</configuration>
```

O Editor do Visual Studio não fornece preenchimento automático de código (IntelliSense) para o `<aws>` elemento ou seus elementos secundários.

Para auxiliá-lo na criação de uma versão corretamente formatada `<aws>`, chame o método `Amazon.AWSConfigs.GenerateConfigTemplate`. Isso gera como saída uma versão canônica do elemento `<aws>` como uma string formatada, que pode ser adaptada para as suas necessidades. As seções a seguir descrevem os atributos e os elementos filho do elemento `<aws>`.

Elementos permitidos

A seguir encontra-se uma lista das relações lógicas entre os elementos permitidos em uma seção de configurações da AWS . Gere a versão mais recente dessa lista chamando o método `Amazon.AWSConfigs.GenerateConfigTemplate`, que gera como saída uma versão canônica do elemento `<aws>` como uma string que pode ser adaptada para as suas necessidades.

```
<aws
  endpointDefinition="string value"
  region="string value"
  profileName="string value"
  profilesLocation="string value">
  <logging
    logTo="None, Log4Net, SystemDiagnostics"
    logResponses="Never | OnError | Always"
    logMetrics="true | false"
    logMetricsFormat="Standard | JSON"
    logMetricsCustomFormatter="NameSpace.Class, Assembly" />
  <dynamoDB
    conversionSchema="V1 | V2">
    <dynamoDBContext
      tableNamePrefix="string value">
      <tableAliases>
```

```
        <alias
          fromTable="string value"
          toTable="string value" />
      </tableAliases>
      <map
        type="NameSpace.Class, Assembly"
        targetTable="string value">
        <property
          name="string value"
          attribute="string value"
          ignore="true | false"
          version="true | false"
          converter="NameSpace.Class, Assembly" />
      </map>
    </dynamoDBContext>
  </dynamoDB>
  <s3
    useSignatureVersion4="true | false" />
  <ec2
    useSignatureVersion4="true | false" />
  <proxy
    host="string value"
    port="1234"
    username="string value"
    password="string value" />
</aws>
```

Referência dos elementos

A seguir está uma lista dos elementos permitidos em uma seção de AWS configurações. Para cada elemento, são listados os atributos permitidos e os elementos filho.

Tópicos

- alias
- aws
- dynamoDB
- dínamo DBContext
- ec2
- registro em log
- mapear

- property
- proxy
- s3

alias

O elemento <alias> representa um único item em um conjunto de um ou mais mapeamentos from-table até to-table, que especifica uma tabela diferente daquela configurada para um tipo. Esse elemento é mapeado para uma instância da classe Amazon.Util.TableAlias da propriedade Amazon.AWSConfigs.DynamoDBConfig.Context.TableAliases no AWS SDK para .NET. O remapeamento é feito antes de aplicar um prefixo de nome da tabela.

Esse elemento pode incluir os seguintes atributos:

fromTable

A parte from-table do mapeamento from-table até to-table. Esse atributo é mapeado para a propriedade Amazon.Util.TableAlias.FromTable no AWS SDK para .NET.

toTable

A parte to-table do mapeamento from-table até to-table. Esse atributo é mapeado para a propriedade Amazon.Util.TableAlias.ToTable no AWS SDK para .NET.

O pai do elemento <alias> é o elemento <tableAliases>.

O elemento <alias> não contém elementos filho.

Veja a seguir um exemplo do elemento <alias> em uso:

```
<alias
  fromTable="Studio"
  toTable="Studios" />
```

aws

O <aws> elemento representa o elemento mais alto em uma seção de AWS configurações. Esse elemento pode incluir os seguintes atributos:

`endpointDefinition`

O caminho absoluto para um arquivo de configuração personalizado que define as AWS regiões e os endpoints a serem usados. Esse atributo é mapeado para a propriedade `Amazon.AWSConfigs.EndpointDefinition` no AWS SDK para .NET.

`profileName`

O nome do perfil das AWS credenciais armazenadas que serão usadas para fazer chamadas de serviço. Esse atributo é mapeado para a propriedade `Amazon.AWSConfigs.AWSProfileName` no AWS SDK para .NET.

`profilesLocation`

O caminho absoluto para a localização do arquivo de credenciais compartilhado com outros AWS SDKs. Por padrão, o arquivo de credenciais está armazenado no diretório `.aws` no diretório base do usuário atual. Esse atributo é mapeado para a propriedade `Amazon.AWSConfigs.AWSProfilesLocation` no AWS SDK para .NET.

`region`

O ID de AWS região padrão para clientes que não especificaram explicitamente uma região. Esse atributo é mapeado para a propriedade `Amazon.AWSConfigs.AWSRegion` no AWS SDK para .NET.

O elemento `<aws>` não possui elemento pai.

O elemento `<aws>` pode incluir os seguintes elementos filho:

- `<dynamoDB>`
- `<ec2>`
- `<logging>`
- `<proxy>`
- `<s3>`

Veja a seguir um exemplo do elemento `<aws>` em uso:

```
<aws
  endpointDefinition="C:\Configs\endpoints.xml"
  region="us-west-2"
```

```
  profileName="development"
  profilesLocation="C:\Configs">
  <!-- ... -->
</aws>
```

dynamoDB

O elemento <dynamoDB> representa um conjunto de configurações do Amazon DynamoDB. Este elemento pode incluir o atributo conversionSchema, que representa a versão a ser usada para conversão entre objetos .NET e DynamoDB. Os valores permitidos incluem V1 e V2. Esse atributo é mapeado para a classe `Amazon.DynamoDBv2.DynamoDBEntryConversion` no AWS SDK para .NET. Para obter mais informações, consulte Série DynamoDB – esquemas de conversão.

O pai do elemento <dynamoDB> é o elemento <aws>.

O elemento <dynamoDB> pode incluir o elemento filho <dynamoDBContext>.

Veja a seguir um exemplo do elemento <dynamoDB> em uso:

```
<dynamoDB
  conversionSchema="V2">
  <!-- ... -->
</dynamoDB>
```

dínamo DBContext

O elemento <dynamoDBContext> representa um conjunto de configurações específicas de contexto do Amazon DynamoDB. Esse elemento pode incluir o tableNamePrefixatributo, que representa o prefixo padrão do nome da tabela que o contexto do DynamoDB usará se não for configurado manualmente. Esse atributo é mapeado para a propriedade `Amazon.Util.DynamoDBContextConfig.TableNamePrefix` da propriedade `Amazon.AWSConfigs.DynamoDBConfig.Context.TableNamePrefix` no AWS SDK para .NET. Para obter mais informações, consulte Aprimoramentos do DynamoDB SDK.

O pai do elemento <dynamoDBContext> é o elemento <dynamoDB>.

O elemento <dynamoDBContext> pode incluir os seguintes elementos filho:

- <alias> (uma ou mais instâncias)
- <map> (uma ou mais instâncias)

Veja a seguir um exemplo do elemento <dynamoDBContext> em uso:

```
<dynamoDBContext
  tableNamePrefix="Test-">
  <!-- ... -->
</dynamoDBContext>
```

ec2

O <ec2> elemento representa uma coleção de EC2 configurações da Amazon. Esse elemento pode incluir o atributo useSignatureVersion4, que especifica se a assinatura da assinatura versão 4 será usada para todas as solicitações (verdadeiro) ou se a assinatura da assinatura versão 4 não será usada para todas as solicitações (falso, o padrão). Esse atributo é mapeado para a propriedade `Amazon.Util.EC2Config.UseSignatureVersion4` da propriedade `Amazon.AWSConfigs.EC2Config.UseSignatureVersion4` no AWS SDK para .NET.

O pai do elemento <ec2> é o elemento.

O elemento <ec2> não contém elementos filho.

Veja a seguir um exemplo do elemento <ec2> em uso:

```
<ec2
  useSignatureVersion4="true" />
```

registro em log

O elemento <logging> representa um conjunto de configurações para o registro de respostas e de métricas de desempenho. Esse elemento pode incluir os seguintes atributos:

`logMetrics`

Se as métricas de desempenho serão registradas para todos os clientes e configurações (verdadeiro); caso contrário, falso. Esse atributo é mapeado para a propriedade `Amazon.Util.LoggingConfig.LogMetrics` da propriedade `Amazon.AWSConfigs.LoggingConfig.LogMetrics` no AWS SDK para .NET.

`logMetricsCustomFormatter`

O tipo de dados e o nome do conjunto de um formatador personalizado para as métricas de registro. Esse atributo é mapeado para a propriedade `Amazon.Util.LoggingConfig.LogMetricsCustomFormatter` da propriedade

`Amazon.AWSConfigs.LoggingConfig.LogMetricsCustomFormatter` no AWS SDK para .NET.

`logMetricsFormat`

O formato em que as métricas de registro são apresentadas (é mapeado para a propriedade `Amazon.Util.LoggingConfig.LogMetricsFormat` da propriedade `Amazon.AWSConfigs.LoggingConfig.LogMetricsFormat` no AWS SDK para .NET).

Os valores permitidos incluem:

`JSON`

Use o formato JSON.

`Standard`

Use o formato padrão.

`logResponses`

Quando registrar as respostas de serviços (é mapeado para a propriedade `Amazon.Util.LoggingConfig.LogResponses` da propriedade `Amazon.AWSConfigs.LoggingConfig.LogResponses` no AWS SDK para .NET).

Os valores permitidos incluem:

`Always`

Sempre registrar as respostas de serviços.

`Never`

Nunca registrar as respostas de serviços.

`OnError`

Somente registrar as respostas de serviços quando existirem erros.

`logTo`

Para onde fazer login (mapeia a `LogTo` propriedade a partir da `Amazon.AWSConfigs.LoggingConfig.LogTo` propriedade no AWS SDK para .NET).

Os valores permitidos incluem um ou mais entre:

`Log4Net`

Registrar no log4net.

`None`

Desativar o registro.

`SystemDiagnostics`

Registrar no System.Diagnostics.

O pai do elemento `<logging>` é o elemento `<aws>`.

O elemento `<logging>` não contém elementos filho.

Veja a seguir um exemplo do elemento `<logging>` em uso:

```
<logging
  logTo="SystemDiagnostics"
  logResponses="OnError"
  logMetrics="true"
  logMetricsFormat="JSON"
  logMetricsCustomFormatter="MyLib.Util.MyMetricsFormatter, MyLib" />
```

mapear

O `<map>` elemento representa um único item em uma coleção de type-to-table mapeamentos de tipos.NET para tabelas do DynamoDB (mapeia para uma instância da `TypeMapping` classe a partir da propriedade no). `Amazon.AWSConfigs.DynamoDBConfig.Context.TypeMappings` AWS SDK para .NET Para obter mais informações, consulte Aprimoramentos do DynamoDB SDK.

Esse elemento pode incluir os seguintes atributos:

`targetTable`

A tabela do DynamoDB a qual o mapeamento se aplica. Esse atributo é mapeado para a propriedade `Amazon.Util.TypeMapping.TargetTable` no AWS SDK para .NET.

`type`

O tipo e o nome do conjunto ao qual o mapeamento se aplica. Esse atributo é mapeado para a propriedade `Amazon.Util.TypeMapping.Type` no AWS SDK para .NET.

O pai do elemento `<map>` é o elemento `<dynamoDBContext>`.

O elemento `<map>` pode incluir uma ou mais instâncias do elemento filho `<property>`.

Veja a seguir um exemplo do elemento <map> em uso:

```
<map
  type="SampleApp.Models.Movie, SampleDLL"
  targetTable="Movies">
  <!-- ... -->
</map>
```

property

O elemento <property> representa uma propriedade do DynamoDB. (Esse elemento é mapeado para uma instância do Amazon.Util. PropertyConfig classe do AddProperty método em AWS SDK para .NET) Para obter mais informações, consulte Aprimoramentos no SDK do DynamoDB e nos atributos do DynamoDB.

Esse elemento pode incluir os seguintes atributos:

attribute

O nome de um atributo da propriedade, como o nome de uma chave de intervalo. Esse atributo é mapeado para a propriedade Amazon.Util.PropertyConfig.Attribute no AWS SDK para .NET.

converter

O tipo de conversor que deve ser usado para esta propriedade. Esse atributo é mapeado para a propriedade Amazon.Util.PropertyConfig.Converter no AWS SDK para .NET.

ignore

Se a propriedade associada deve ser ignorada (verdadeiro); caso contrário, falso. Esse atributo é mapeado para a propriedade Amazon.Util.PropertyConfig.Ignore no AWS SDK para .NET.

name

O nome da propriedade. Esse atributo é mapeado para a propriedade Amazon.Util.PropertyConfig.Name no AWS SDK para .NET.

version

Se essa propriedade deve armazenar o número da versão do item (verdadeiro); caso contrário, falso. Esse atributo é mapeado para a propriedade Amazon.Util.PropertyConfig.Version no AWS SDK para .NET.

O pai do elemento <property> é o elemento <map>.

O elemento <property> não contém elementos filho.

Veja a seguir um exemplo do elemento <property> em uso:

```
<property
  name="Rating"
  converter="SampleApp.Models.RatingConverter, SampleDLL" />
```

proxy

O elemento <proxy> representa as configurações necessárias de um proxy para o seu uso pelo AWS SDK para .NET . Esse elemento pode incluir os seguintes atributos:

host

O nome do host ou endereço IP do servidor de proxy. Esses atributos são mapeados para a propriedade `Amazon.Util.ProxyConfig.Host` da propriedade `Amazon.AWSConfigs.ProxyConfig.Host` no AWS SDK para .NET.

password

A senha para a autenticação com o servidor de proxy. Esses atributos são mapeados para a propriedade `Amazon.Util.ProxyConfig.Password` da propriedade `Amazon.AWSConfigs.ProxyConfig.Password` no AWS SDK para .NET.

porta

O número da porta do proxy. Esses atributos são mapeados para a propriedade `Amazon.Util.ProxyConfig.Port` da propriedade `Amazon.AWSConfigs.ProxyConfig.Port` no AWS SDK para .NET.

username

O nome de usuário para a autenticação com o servidor de proxy. Esses atributos são mapeados para a propriedade `Amazon.Util.ProxyConfig.Username` da propriedade `Amazon.AWSConfigs.ProxyConfig.Username` no AWS SDK para .NET.

O pai do elemento <proxy> é o elemento <aws>.

O elemento <proxy> não contém elementos filho.

Veja a seguir um exemplo do elemento <proxy> em uso:

```
<proxy
  host="192.0.2.0"
  port="1234"
  username="My-Username-Here"
  password="My-Password-Here" />
```

s3

O elemento <s3> representa um conjunto de configurações do Amazon S3. Esse elemento pode incluir o atributo useSignatureVersion4, que especifica se a assinatura da assinatura versão 4 será usada para todas as solicitações (verdadeiro) ou se a assinatura da assinatura versão 4 não será usada para todas as solicitações (falso, o padrão). Esse atributo é mapeado para a propriedade `Amazon.AWSConfigs.S3Config.UseSignatureVersion4` no AWS SDK para .NET.

O pai do elemento <s3> é o elemento <aws>.

O elemento <s3> não contém elementos filho.

Veja a seguir um exemplo do elemento <s3> em uso:

```
<s3 useSignatureVersion4="true" />
```

Uso de credenciais herdadas

Os tópicos desta seção fornecem informações sobre como usar credenciais de longo ou curto prazo sem usar o Centro de Identidade do AWS IAM.

> **Warning**
>
> Para evitar riscos de segurança, não use usuários do IAM para autenticação ao desenvolver software com propósito específico ou trabalhar com dados reais. Em vez disso, use federação com um provedor de identidade, como Centro de Identidade do AWS IAM.

> **Note**
>
> As informações neste tópico são para circunstâncias em que você precisa obter e gerenciar manualmente as credenciais de curto ou longo prazo. Para obter informações adicionais

sobre credenciais de curto e longo prazo, consulte Outras formas de autenticação no Guia de referência de ferramentas AWS SDKs e ferramentas.
Para obter as melhores práticas de segurança Centro de Identidade do AWS IAM, use, conforme descrito emAutenticando com AWS.

Avisos e orientações importantes para credenciais

Avisos para credenciais

- NÃO use as credenciais raiz da sua conta para acessar AWS recursos. Estas credenciais fornecem acesso ilimitado à conta e são difíceis de revogar.
- NÃO coloque chaves de acesso literais ou informações de credenciais em arquivos de aplicações. Se colocar, criará um risco de exposição acidental das credenciais se, por exemplo, fizer upload do projeto em um repositório público.
- NÃO inclua arquivos que contenham credenciais em sua área de projeto.
- Esteja ciente de que todas as credenciais armazenadas no AWS `credentials` arquivo compartilhado são armazenadas em texto simples.

Orientação adicional para gerenciar credenciais com segurança

Para uma discussão geral sobre como gerenciar AWS credenciais com segurança, consulte Credenciais de AWS segurança em e Melhores práticas de segurança Referência geral da AWSe casos de uso no Guia do usuário do IAM. Além dessas discussões, considere o seguinte:

- Crie usuários adicionais, como usuários no Centro de Identidade do IAM, e use as credenciais deles em vez de usar suas credenciais de usuário raiz da AWS . As credenciais de outros usuários poderão ser revogadas, se necessário, ou são temporárias por natureza. Além disso, você pode aplicar uma política a cada usuário para acessar somente determinados recursos e ações e, assim, adotar uma postura de permissões com privilégio mínimo.

- Use perfis do IAM para tarefas do Amazon Elastic Container Service (Amazon ECS).

- Use perfis do IAM para aplicações em execução nas instâncias do Amazon EC2.

- Use credenciais temporárias ou variáveis de ambiente para aplicativos disponíveis para usuários de fora da organização.

Tópicos

- Usando o arquivo de AWS credenciais compartilhado
- Usando a SDK Store (somente Windows)

Usando o arquivo de AWS credenciais compartilhado

(Assegure-se de revisar os avisos e orientações importantes para credenciais.)

Uma forma de fornecer credenciais para seus aplicativos é criar perfis no arquivo de credenciais compartilhado da AWS e, em seguida, armazenar as credenciais nesses perfis. Esse arquivo pode ser usado pelo outro AWS SDKs. Ele também pode ser usado pelo AWS CLIAWS Tools for Windows PowerShell, pelo e pelos AWS kits de ferramentas do Visual Studio e do VS Code. JetBrains

> **Warning**
>
> Para evitar riscos de segurança, não use usuários do IAM para autenticação ao desenvolver software com propósito específico ou trabalhar com dados reais. Em vez disso, use federação com um provedor de identidade, como Centro de Identidade do AWS IAM.

> **Note**
>
> As informações neste tópico são para circunstâncias em que você precisa obter e gerenciar manualmente as credenciais de curto ou longo prazo. Para obter informações adicionais sobre credenciais de curto e longo prazo, consulte Outras formas de autenticação no Guia de referência de ferramentas AWS SDKs e ferramentas.
> Para obter as melhores práticas de segurança Centro de Identidade do AWS IAM, use, conforme descrito emAutenticando com AWS.

Informações gerais

Por padrão, o arquivo de AWS credenciais compartilhadas está localizado no `.aws` diretório dentro do seu diretório inicial e é nomeado`credentials`; ou seja, `~/.aws/credentials` (Linux ou

macOS) `%USERPROFILE%\.aws\credentials` ou (Windows). Para obter informações sobre locais alternativos, consulte Localização dos arquivos compartilhados no Guia de referência de ferramentas AWS SDKs e ferramentas. Também consulte Acessar credenciais e perfis em um aplicativo.

O arquivo de AWS credenciais compartilhado é um arquivo de texto simples e segue um determinado formato. Para obter informações sobre o formato dos arquivos de AWS credenciais, consulte Formato do arquivo de credenciais no Guia de Referência de Ferramentas AWS SDKs e Ferramentas.

Você pode gerenciar os perfis no arquivo de AWS credenciais compartilhadas de várias maneiras.

- Use qualquer editor de texto para criar e atualizar o arquivo de AWS credenciais compartilhado.
- Use o Amazon.Runtime. CredentialManagementnamespace da AWS SDK para .NET API, conforme mostrado posteriormente neste tópico.
- Use comandos e procedimentos para o Ferramentas da AWS para PowerShelle os AWS kits de ferramentas para Visual Studio e VS Code. JetBrains
- Usar comandos do AWS CLI; por exemplo, `aws configure set aws_access_key_id` e `aws configure set aws_secret_access_key`.

Exemplos de gerenciamento de perfil

As seções a seguir mostram exemplos de perfis no arquivo de AWS credenciais compartilhado. Alguns dos exemplos mostram o resultado, que pode ser obtido através de qualquer um dos métodos de gerenciamento de credenciais descritos anteriormente. Outros exemplos mostram como usar um método específico.

O perfil padrão.

O arquivo de AWS credenciais compartilhado quase sempre terá um perfil chamado default. É aqui que ele AWS SDK para .NET procura as credenciais se nenhum outro perfil estiver definido.

O perfil `[default]` geralmente é parecido com o mostrado a seguir.

```
[default]
aws_access_key_id = AKIAIOSFODNN7EXAMPLE
aws_secret_access_key = wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
```

Criar um perfil de forma programática

Este exemplo mostra como criar um perfil e salvá-lo programaticamente no arquivo de AWS credenciais compartilhadas. Ele usa as seguintes classes do Amazon.Runtime.CredentialManagementnamespace: CredentialProfileOptionsCredentialProfile, e. SharedCredentialsFile

```
using Amazon.Runtime.CredentialManagement;
...

// Do not include credentials in your code.
WriteProfile("my_new_profile", SecurelyStoredKeyID, SecurelyStoredSecretAccessKey);
...

void WriteProfile(string profileName, string keyId, string secret)
{
    Console.WriteLine($"Create the [{profileName}] profile...");
    var options = new CredentialProfileOptions
    {
        AccessKey = keyId,
        SecretKey = secret
    };
    var profile = new CredentialProfile(profileName, options);
    var sharedFile = new SharedCredentialsFile();
    sharedFile.RegisterProfile(profile);
}
```

> **Warning**
>
> Códigos como esse geralmente não deveriam estar em seu aplicativo. Se você inclui-lo em seu aplicativo, tome as devidas precauções para garantir que as chaves de texto simples não possam ser vistas no código, na rede ou mesmo na memória do computador.

A seguir está o perfil que foi criado por este exemplo.

```
[my_new_profile]
aws_access_key_id=AKIAIOSFODNN7EXAMPLE
aws_secret_access_key=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
```

Atualizar um perfil existente de forma programática

Este exemplo mostra como atualizar de forma programática o perfil que foi criado anteriormente. Ele usa as seguintes classes do Amazon.Runtime. CredentialManagementnamespace: e. CredentialProfileSharedCredentialsFile Ele também usa a RegionEndpointclasse do namespace Amazon.

```
using Amazon.Runtime.CredentialManagement;
...

AddRegion("my_new_profile", RegionEndpoint.USWest2);
...

void AddRegion(string profileName, RegionEndpoint region)
{
    var sharedFile = new SharedCredentialsFile();
    CredentialProfile profile;
    if (sharedFile.TryGetProfile(profileName, out profile))
    {
        profile.Region = region;
        sharedFile.RegisterProfile(profile);
    }
}
```

A seguir está o perfil atualizado.

```
[my_new_profile]
aws_access_key_id=AKIAIOSFODNN7EXAMPLE
aws_secret_access_key=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
region=us-west-2
```

> **Note**
>
> Você também pode definir a AWS região em outros locais e usando outros métodos. Para obter mais informações, consulte Definindo a AWS região para o AWS SDK para .NET.

Usando a SDK Store (somente Windows)

(Assegure-se de revisar os avisos e diretrizes importantes.)

No Windows, a SDK Store é outro local para criar perfis e armazenar credenciais criptografadas para seu AWS SDK para .NET aplicativo. Ela está localizada em `%USERPROFILE%\AppData\Local\AWSToolkit\RegisteredAccounts.json`. Você pode usar a SDK Store durante o desenvolvimento como alternativa ao arquivo de credenciais compartilhado da AWS.

> **Warning**
>
> Para evitar riscos de segurança, não use usuários do IAM para autenticação ao desenvolver software com propósito específico ou trabalhar com dados reais. Em vez disso, use federação com um provedor de identidade, como Centro de Identidade do AWS IAM.

> **Note**
>
> As informações neste tópico são para circunstâncias em que você precisa obter e gerenciar manualmente as credenciais de curto ou longo prazo. Para obter informações adicionais sobre credenciais de curto e longo prazo, consulte Outras formas de autenticação no Guia de referência de ferramentas AWS SDKs e ferramentas.
> Para obter as melhores práticas de segurança Centro de Identidade do AWS IAM, use, conforme descrito emAutenticando com AWS.

Informações gerais

A SDK Store oferece os seguintes benefícios:

- As credenciais na SDK Store são criptografadas, e a SDK Store reside no diretório base do usuário. Isso limita o risco de exposição acidental das credenciais.
- A SDK Store também oferece credenciais para o AWS Tools for Windows PowerShell e o AWS Toolkit for Visual Studio.

Os perfis da SDK Store são específicos para um determinado usuário em um host particular. Não é possível copiá-los para outros hosts ou outros usuários. Isso significa que não é possível reutilizar os perfis da SDK Store que estiverem na máquina de desenvolvimento para outros hosts ou outras máquinas de desenvolvedores. Isso também significa que você não pode usar perfis da SDK Store em aplicativos de produção.

Gerencie os perfis na SDK Store das seguintes formas:

- Usar a interface gráfica de usuário (GUI) no AWS Toolkit for Visual Studio.
- Use o Amazon.Runtime. CredentialManagementnamespace da AWS SDK para .NET API, conforme mostrado posteriormente neste tópico.
- Usar comandos do AWS Tools for Windows PowerShell; por exemplo, `Set-AWSCredential` e `Remove-AWSCredentialProfile`.

Exemplos de gerenciamento de perfil

Os exemplos a seguir mostram como criar e atualizar um perfil de forma programática na SDK Store.

Criar um perfil de forma programática

Este exemplo mostra como criar um perfil e salvá-lo na SDK Store de forma programática.
Ele usa as seguintes classes do Amazon.Runtime. CredentialManagementnamespace:
CredentialProfileOptionsCredentialProfile, e Net SDKCredentials File.

```
using Amazon.Runtime.CredentialManagement;
...

// Do not include credentials in your code.
WriteProfile("my_new_profile", SecurelyStoredKeyID, SecurelyStoredSecretAccessKey);
...

void WriteProfile(string profileName, string keyId, string secret)
{
    Console.WriteLine($"Create the [{profileName}] profile...");
    var options = new CredentialProfileOptions
    {
        AccessKey = keyId,
        SecretKey = secret
    };
    var profile = new CredentialProfile(profileName, options);
    var netSdkStore = new NetSDKCredentialsFile();
    netSdkStore.RegisterProfile(profile);
}
```

> **Warning**
>
> Códigos como esse geralmente não deveriam estar em seu aplicativo. Se ele estiver incluído em seu aplicativo, tome as devidas precauções para garantir que as chaves de texto simples não possam ser vistas no código, na rede ou mesmo na memória do computador.

A seguir está o perfil que foi criado por este exemplo.

```
"[generated GUID]" : {
    "AWSAccessKey" : "01000000D08...[etc., encrypted access key ID]",
    "AWSSecretKey" : "01000000D08...[etc., encrypted secret access key]",
    "ProfileType"  : "AWS",
    "DisplayName"  : "my_new_profile",
}
```

Atualizar um perfil existente de forma programática

Este exemplo mostra como atualizar de forma programática o perfil que foi criado anteriormente. Ele usa as seguintes classes do Amazon.Runtime. CredentialManagementnamespace: CredentialProfilee Net SDKCredentials File. Ele também usa a RegionEndpointclasse do namespace Amazon.

```
using Amazon.Runtime.CredentialManagement;
...

AddRegion("my_new_profile", RegionEndpoint.USWest2);
...

void AddRegion(string profileName, RegionEndpoint region)
{
    var netSdkStore = new NetSDKCredentialsFile();
    CredentialProfile profile;
    if (netSdkStore.TryGetProfile(profileName, out profile))
    {
        profile.Region = region;
        netSdkStore.RegisterProfile(profile);
    }
}
```

A seguir está o perfil atualizado.

```
"[generated GUID]" : {
    "AWSAccessKey" : "01000000D08...[etc., encrypted access key ID]",
    "AWSSecretKey" : "01000000D08...[etc., encrypted secret access key]",
    "ProfileType"  : "AWS",
    "DisplayName"  : "my_new_profile",
    "Region"       : "us-west-2"
}
```

> **Note**
>
> Você também pode definir a AWS região em outros locais e usando outros métodos. Para obter mais informações, consulte Definindo a AWS região para o AWS SDK para .NET.

Usando o AWS SDK para .NET

Esta seção fornece informações sobre os recursos do AWS SDK para .NET que talvez você precise considerar ao criar seus aplicativos.

Antes de começar, certifique-se de ter configurado seu ambiente e seu projeto.

Para obter informações sobre o desenvolvimento de software para AWS serviços específicos, juntamente com exemplos de código, consulte AWS Serviços de chamadas. Consulte mais exemplos de código em SDK para .NET (v4) exemplos de código.

Tópicos

- Programação assíncrona usando o AWS SDK para .NET
- Usando resultados paginados no AWS SDK para .NET
- Support para HTTP 2 no AWS SDK para .NET
- Ferramentas adicionais

Programação assíncrona usando o AWS SDK para .NET

O AWS SDK para .NET usa o Padrão Assíncrono Baseado em Tarefas (TAP) para sua implementação assíncrona. Para saber mais sobre o TAP, consulte Padrão assíncrono baseado em tarefas (TAP) em docs.microsoft.com.

Este tópico fornece uma visão geral de como usar o TAP em suas chamadas para clientes AWS de atendimento.

Os métodos assíncronos na AWS SDK para .NET API são operações baseadas na `Task` classe ou na classe. `Task<TResult>` Consulte docs.microsoft.com para obter informações sobre essas classes: classe Task, Task< > class. TResult

Quando esses métodos de API são chamados em seu código, eles devem ser chamados em uma função declarada com a palavra-chave `async`, conforme mostrado no exemplo a seguir.

```
static async Task Main(string[] args)
{
  ...
  // Call the function that contains the asynchronous API method.
  // Could also call the asynchronous API method directly from Main
```

```
  //  because Main is declared async
  var response = await ListBucketsAsync();
  Console.WriteLine($"Number of buckets: {response.Buckets.Count}");
  ...
}

// Async method to get a list of Amazon S3 buckets.
private static async Task<ListBucketsResponse> ListBucketsAsync()
{
  ...
  var response = await s3Client.ListBucketsAsync();
  return response;
}
```

Conforme mostrado no trecho de código anterior, o escopo preferencial para a declaração `async` é a função `Main`. Definir esse escopo `async` garante que todas as chamadas para clientes de serviço da AWS sejam obrigatoriamente assíncronas. Se não puder declarar que `Main` seja assíncrono por algum motivo, você poderá usar a palavra-chave `async` em outras funções de `Main` e, em seguida, chamar os métodos da API a partir daí, conforme mostrado no exemplo a seguir.

```
static void Main(string[] args)
{
  ...
  Task<ListBucketsResponse> response = ListBucketsAsync();
  Console.WriteLine($"Number of buckets: {response.Result.Buckets.Count}");
  ...
}

// Async method to get a list of Amazon S3 buckets.
private static async Task<ListBucketsResponse> ListBucketsAsync()
{
  ...
  var response = await s3Client.ListBucketsAsync();
  return response;
}
```

Observe a sintaxe especial `Task<>` que é necessária em `Main` quando você usa esse padrão. Além disso, você deve usar o membro **`Result`** da resposta para obter os dados.

Você pode ver exemplos completos de chamadas assíncronas para clientes AWS de serviço na Criando um aplicativo simples seção (Aplicativo simples para várias plataformaseAplicativo simples baseado no Windows) e em. Exemplos de código guiado

Usando resultados paginados no AWS SDK para .NET

Alguns AWS serviços coletam e armazenam uma grande quantidade de dados, que você pode recuperar usando as chamadas de API do AWS SDK para .NET. Se a quantidade de dados que você deseja recuperar se tornar muito grande para uma única chamada de API, você poderá dividir os resultados em partes mais gerenciáveis usando a paginação.

Para permitir que você realize a paginação, os objetos de solicitação e resposta de muitos clientes de serviço no SDK fornecem um token de continuação (normalmente intitulado `NextToken`). Alguns desses clientes de serviço também fornecem paginadores.

Os paginadores permitem que você evite a sobrecarga do token de continuação, que pode envolver loops, variáveis de estado, múltiplas chamadas de API e assim por diante. Ao usar um paginador, você pode recuperar dados de um serviço da AWS através de uma única linha de código, a declaração de um loop `foreach`. Se forem necessárias múltiplas chamadas de API para recuperar os dados, o paginador lidará com isso para você.

Onde posso encontrar paginadores?

Nem todos os serviços oferecem paginadores. Uma forma de determinar se um serviço fornece um paginador para uma API específica é examinar a definição de uma classe de cliente de serviço na Referência de API do AWS SDK para .NET.

Por exemplo, se você examinar a definição da AmazonCloudWatchLogsClientclasse, verá uma `Paginators` propriedade. Essa é a propriedade que fornece um paginador para o Amazon CloudWatch Logs.

O que os paginadores me oferecem?

Os paginadores contêm propriedades que permitem que você veja as respostas completas. Eles também costumam conter uma ou mais propriedades que permitem acessar as partes mais interessantes das respostas, que chamaremos de resultados-chave.

Por exemplo, no `AmazonCloudWatchLogsClient` mencionado anteriormente, o `Paginator` objeto contém uma `Responses` propriedade com o DescribeLogGroupsResponseobjeto completo da chamada da API. Essa propriedade `Responses` contém, entre outras coisas, uma coleção dos grupos de logs.

O objeto Paginator também contém um resultado chave chamado `LogGroups`. Essa propriedade contém apenas a parte da resposta dos grupos de logs. Ter esse resultados-chaves permite que você reduza e simplifique seu código em muitas circunstâncias.

Paginação síncrona em comparação à assíncrona

Os paginadores fornecem mecanismos síncronos e assíncronos para paginação. A paginação síncrona está disponível em projetos do.NET Framework 4.7.2 (ou posterior). A paginação assíncrona está disponível em projetos do.NET Core (.NET Core 3.1, .NET 5 e assim por diante).

Como as operações assíncronas e o .NET Core são recomendadas, o exemplo a seguir mostra a paginação assíncrona. As informações sobre como realizar as mesmas tarefas usando paginação síncrona e o.NET Framework 4.7.2 (ou posterior) são mostradas após o exemplo em. Considerações adicionais sobre paginadores

Exemplo

O exemplo a seguir mostra como usar o AWS SDK para .NET para exibir uma lista de grupos de registros. Para fins de comparação, o exemplo mostra como fazer isso com e sem paginadores. Antes de examinar o código completo, mostrado posteriormente, considere os trechos a seguir.

Obtendo grupos CloudWatch de registros sem paginadores

```
      // Loop as many times as needed to get all the log groups
      var request = new DescribeLogGroupsRequest{Limit = LogGroupLimit};
      do
      {
        Console.WriteLine($"Getting up to {LogGroupLimit} log groups...");
        var response = await cwClient.DescribeLogGroupsAsync(request);
        foreach(var logGroup in response.LogGroups)
        {
          Console.WriteLine($"{logGroup.LogGroupName}");
        }
        request.NextToken = response.NextToken;
      } while(!string.IsNullOrEmpty(request.NextToken));
```

Obtendo grupos de CloudWatch registros usando paginadores

```
      // No need to loop to get all the log groups--the SDK does it for us behind the
  scenes
```

```
    var paginatorForLogGroups =
      cwClient.Paginators.DescribeLogGroups(new DescribeLogGroupsRequest());
    await foreach(var logGroup in paginatorForLogGroups.LogGroups)
    {
      Console.WriteLine(logGroup.LogGroupName);
    }
```

Os resultados desses dois trechos são exatamente os mesmos, então a vantagem de usar paginadores pode ser vista claramente.

> **Note**
>
> Antes de tentar criar e executar o código completo, certifique-se de ter configurado seu ambiente e projeto.
> Talvez você também precise do Microsoft.Bcl. AsyncInterfaces NuGet pacote porque paginadores assíncronos usam a interface. `IAsyncEnumerable`

Código completo

Esta seção mostra as referências relevantes e o código completo desse exemplo.

Referências do SDK

NuGet pacotes:

- AWSSDK.CloudWatch

Elementos de programação:

- Namespace Amazon. CloudWatch

  Classe AmazonCloudWatchLogsClient
- Namespace Amazon. CloudWatchLogs.Modelo

  Classe DescribeLogGroupsRequest

  Classe DescribeLogGroupsResponse

  Classe LogGroup

Código completo

```
using System;
using System.Threading.Tasks;
using Amazon.CloudWatchLogs;
using Amazon.CloudWatchLogs.Model;

namespace CWGetLogGroups
{
  class Program
  {
    // A small limit for demonstration purposes
    private const int LogGroupLimit = 3;

    //
    // Main method
    static async Task Main(string[] args)
    {
      var cwClient = new AmazonCloudWatchLogsClient();
      await DisplayLogGroupsWithoutPaginators(cwClient);
      await DisplayLogGroupsWithPaginators(cwClient);
    }


    //
    // Method to get CloudWatch log groups without paginators
    private static async Task DisplayLogGroupsWithoutPaginators(IAmazonCloudWatchLogs
 cwClient)
    {
      Console.WriteLine("\nGetting list of CloudWatch log groups without using
 paginators...");

 Console.WriteLine("-----------------------------------------------------------------");

      // Loop as many times as needed to get all the log groups
      var request = new DescribeLogGroupsRequest{Limit = LogGroupLimit};
      do
      {
        Console.WriteLine($"Getting up to {LogGroupLimit} log groups...");
        DescribeLogGroupsResponse response = await
 cwClient.DescribeLogGroupsAsync(request);
        foreach(LogGroup logGroup in response.LogGroups)
        {
          Console.WriteLine($"{logGroup.LogGroupName}");
```

```
        }
        request.NextToken = response.NextToken;
      } while(!string.IsNullOrEmpty(request.NextToken));
    }


    //
    // Method to get CloudWatch log groups by using paginators
    private static async Task DisplayLogGroupsWithPaginators(IAmazonCloudWatchLogs
cwClient)
    {
      Console.WriteLine("\nGetting list of CloudWatch log groups by using
paginators...");

Console.WriteLine("------------------------------------------------------------");

      // Access the key results; i.e., the log groups
      // No need to loop to get all the log groups--the SDK does it for us behind the
scenes
      Console.WriteLine("\nFrom the key results...");
      Console.WriteLine("------------------------");
      IDescribeLogGroupsPaginator paginatorForLogGroups =
        cwClient.Paginators.DescribeLogGroups(new DescribeLogGroupsRequest());
      await foreach(LogGroup logGroup in paginatorForLogGroups.LogGroups)
      {
        Console.WriteLine(logGroup.LogGroupName);
      }

      // Access the full response
      // Create a new paginator, do NOT reuse the one from above
      Console.WriteLine("\nFrom the full response...");
      Console.WriteLine("-------------------------");
      IDescribeLogGroupsPaginator paginatorForResponses =
        cwClient.Paginators.DescribeLogGroups(new DescribeLogGroupsRequest());
      await foreach(DescribeLogGroupsResponse response in
paginatorForResponses.Responses)
      {
        Console.WriteLine($"Content length: {response.ContentLength}");
        Console.WriteLine($"HTTP result: {response.HttpStatusCode}");
        Console.WriteLine($"Metadata: {response.ResponseMetadata}");
        Console.WriteLine("Log groups:");
        foreach(LogGroup logGroup in response.LogGroups)
        {
          Console.WriteLine($"\t{logGroup.LogGroupName}");
```

```
        }
      }
    }
  }
}
```

Considerações adicionais sobre paginadores

- Os paginadores não podem ser usados mais de uma vez

  Se você precisar dos resultados de um AWS paginador específico em vários locais em seu código, você não deve usar um objeto paginador mais de uma vez. Em vez disso, crie um novo paginador sempre que precisar dele. Esse conceito é mostrado no exemplo de código anterior no método `DisplayLogGroupsWithPaginators`.

- Paginação síncrona

  A paginação síncrona está disponível para projetos do.NET Framework 4.7.2 (ou posterior).

  Para ver isso, crie um projeto do.NET Framework 4.7.2 (ou posterior) e copie o código anterior para ele. Em seguida, basta remover a palavra-chave `await` das duas chamadas do paginador `foreach`, conforme mostrado no exemplo a seguir.

  ```
  /*await*/ foreach(var logGroup in paginatorForLogGroups.LogGroups)
  {
    Console.WriteLine(logGroup.LogGroupName);
  }
  ```

  Crie e execute o projeto para ver os mesmos resultados que você viu com a paginação assíncrona.

Support para HTTP 2 no AWS SDK para .NET

Alguns AWS serviços e operações exigem HTTP 2. Por exemplo, o streaming bidirecional no Amazon Transcribe Streaming não é possível via HTTP 1.1 e, portanto, requer HTTP 2. A versão 4 do suporte AWS SDK para .NET adicional para HTTP 2 para que você possa usar essas operações em seus aplicativos. Para uma operação HTTP 2 bidirecional, o comportamento do SDK ao receber

fluxos é semelhante ao do HTTP 1.1. Ou seja, quando aplicativos que usam o SDK enviam eventos para o serviço, a solicitação tem um editor atribuído pelo desenvolvedor.

Para ver esse comportamento em ação, considere o exemplo a seguir para o Amazon Transcribe Streaming. Ele usa a Amazon. TranscribeStreaminge Amazon. TranscribeStreaming.Espaços de nomes do modelo.

Neste exemplo, o desenvolvedor define a `StartStreamTranscriptionRequest.AudioStreamPublisher` propriedade com uma função de retorno de chamada, que é um.NET`Func`. O SDK usa o for `Func` definido `AudioStreamPublisher` para extrair eventos do código do usuário para transmitir para o usuário. O SDK chama o `Func` até que ele retorne nulo.

O código demonstra como o áudio de um arquivo pode ser transmitido para o Amazon Transcribe Streaming para processamento.

```
using Amazon;
using Amazon.TranscribeStreaming;
using Amazon.TranscribeStreaming.Model;

CancellationTokenSource cancelSource = new CancellationTokenSource();

var client = new AmazonTranscribeStreamingClient(RegionEndpoint.USEast1);

var startRequest = new StartStreamTranscriptionRequest
{
    LanguageCode = LanguageCode.EnUS,
    MediaEncoding = MediaEncoding.Flac,
    MediaSampleRateHertz = 44100,
    NumberOfChannels = 2,
    EnableChannelIdentification = true
};

Stream fileStream = File.OpenRead("hello-world.flac");
var buffer = new byte[1024 * 10];
startRequest.AudioStreamPublisher += async () =>
{
    var bytesRead = await fileStream.ReadAsync(buffer, 0, buffer.Length);

    if (bytesRead == 0)
        return null;
```

```
    var audioEvent = new AudioEvent
    {
        AudioChunk = new MemoryStream(buffer, 0, bytesRead)
    };

    return audioEvent;
};

using var response = await client.StartStreamTranscriptionAsync(startRequest);
Console.WriteLine(response.HttpStatusCode);

response.TranscriptResultStream.ExceptionReceived +=
 TranscriptResultStream_ExceptionReceived;
response.TranscriptResultStream.TranscriptEventReceived +=
 TranscriptResultStream_TranscriptEventReceived;

void TranscriptResultStream_ExceptionReceived(object? sender,
 Amazon.Runtime.EventStreams.EventStreamExceptionReceivedArgs<TranscribeStreamingEventStreamExc
 e)
{
    Console.WriteLine(e.EventStreamException.Message);
    cancelSource.Cancel();
}
void TranscriptResultStream_TranscriptEventReceived(object? sender,
 Amazon.Runtime.EventStreams.EventStreamEventReceivedArgs<TranscriptEvent> e)
{
    foreach (var result in e.EventStreamEvent.Transcript.Results)
    {
        if (!string.Equals("ch_0", result.ChannelId,
 StringComparison.OrdinalIgnoreCase))
            continue;

        var text = result.Alternatives[0].Transcript;
        if (!string.IsNullOrEmpty(text))
        {
            Console.WriteLine(text);
        }
    }
}

_ = response.TranscriptResultStream.StartProcessingAsync();

try
{
```

```
    await Task.Delay(10000, cancelSource.Token);
}
catch (TaskCanceledException) { }
```

> **Warning**
>
> Algumas operações HTTP 2 bidirecionais, como o método `InvokeModelWithBidirectionalStreamAsync` da Amazon Bedrock, a Amazon.BedrockRuntimenamespace, não retorne uma resposta da invocação inicial no cliente de serviço até que alguns eventos tenham sido publicados. Esse comportamento pode fazer com que seu aplicativo seja bloqueado. Para evitar essa situação, separe o código do aplicativo que fornece eventos ao editor e execute-o em um encadeamento diferente do encadeamento que invoca as operações no cliente do serviço.

Considerações adicionais

- AWS SDK para .NET o suporte para HTTP 2 está disponível somente em versões voltadas para o.NET 8 e versões posteriores. Ele não está disponível em versões voltadas para o.NET Framework.
- Para obter informações mais detalhadas, consulte PR 3730 no aws-sdk-net GitHubrepositório.

Ferramentas adicionais

A seguir estão algumas ferramentas adicionais que você pode usar para facilitar o trabalho de desenvolvimento, implantação e manutenção de seus aplicativos .NET.

AWS Ferramenta de implantação

Depois de desenvolver seu aplicativo.NET Core nativo em nuvem em uma máquina de desenvolvimento, você pode usar a Ferramenta de Implantação para a CLI do.NET para AWS implantar seu aplicativo com mais facilidade em. AWS

Para obter mais informações, consulte Implemente aplicativos em AWS.

AWS Estrutura de processamento de mensagens para.NET

Se você estiver usando serviços como Amazon SQS, Amazon SNS ou EventBridge Amazon, poderá aproveitar a estrutura de processamento AWS de mensagens para.NET. Para obter mais informações, consulte AWS Estrutura de processamento de mensagens para.NET.

Integrações com o.NET Aspire

Você pode aproveitar as vantagens das integrações com o.NET Aspire para melhorar o ciclo interno de desenvolvimento. Para obter mais informações, consulte Integração AWS com o.NET Aspire no AWS SDK para .NET.

Autenticação e autorização avançadas com o AWS SDK para .NET

Os tópicos desta seção fornecem informações sobre técnicas avançadas de autenticação e autorização em seu AWS SDK para .NET aplicativo.

Tópicos

- Login único com o AWS SDK para .NET

Login único com o AWS SDK para .NET

Centro de Identidade do AWS IAM é um serviço de login único (SSO) baseado em nuvem que facilita o gerenciamento centralizado do acesso por SSO a todos os seus aplicativos e à nuvem. Contas da AWS Para obter detalhes completos, consulte o Guia do usuário do Centro de Identidade do IAM.

Se você não estiver familiarizado com a forma como um SDK interage com o Centro de Identidade do IAM, consulte as informações a seguir.

Padrão de interação de alto nível

Em um alto nível, SDKs interaja com o IAM Identity Center de maneira semelhante ao seguinte padrão:

1. O Centro de Identidade do IAM é configurado, normalmente por meio do console do Centro de Identidade do IAM e um usuário de SSO é convidado a participar.
2. O AWS `config` arquivo compartilhado no computador do usuário é atualizado com informações de SSO.
3. O usuário faz login por meio do IAM Identity Center e recebe credenciais de curto prazo para as permissões AWS Identity and Access Management (IAM) que foram configuradas para ele. Esse login pode ser iniciado por meio de uma ferramenta que não é do SDK, como o AWS CLI, ou programaticamente por meio de um aplicativo.NET.
4. O usuário continua fazendo seu trabalho. Quando outros aplicativos que usam SSO são executados, não é necessário logar novamente para abrir os aplicativos.

O restante deste tópico fornece informações de referência para configuração e uso de Centro de Identidade do AWS IAM. Ele fornece informações complementares e mais avançadas do que a

configuração básica do SSO em Autenticando com AWS. Se você é iniciante no SSO em AWS, talvez queira examinar esse tópico primeiro para obter informações fundamentais e, em seguida, os seguintes tutoriais para ver o SSO em ação:

- Tutorial: somente aplicativo .NET
- Tutorial: AWS CLI e aplicativo.NET

Este tópico contém as seguintes seções:

- Pré-requisitos
- Como configurar um perfil de SSO
- Como gerar e usar de tokens de SSO
- Recursos adicionais
- Tutoriais

Pré-requisitos

Antes de usar o IAM Identity Center, você deve realizar determinadas tarefas, como escolher uma fonte de identidade e configurar os aplicativos relevantes Contas da AWS . Para obter informações adicionais, consulte:

- Para obter informações gerais sobre essas tarefas, consulte Conceitos básicos no Guia do usuário do Centro de Identidade do IAM
- Para obter exemplos de tarefas específicas, consulte a lista de tutoriais no final deste tópico. Contudo, certifique-se de revisar as informações neste tópico antes de experimentar os tutoriais.

Como configurar um perfil de SSO

Depois que o IAM Identity Center for configurado no local relevante Conta da AWS, um perfil nomeado para SSO deve ser adicionado ao AWS `config` arquivo compartilhado do usuário. Esse perfil é usado para se conectar ao portal de acesso da AWS, que retorna credenciais de curto prazo para as permissões do IAM que foram configuradas para o usuário.

O arquivo `config` compartilhado geralmente é nomeado `%USERPROFILE%\.aws\config` no Windows e `~/.aws/config` no Linux e no macOS. Você pode usar seu editor de texto preferido para adicionar um novo perfil para SSO. Como alternativa, você pode usar o comando `aws`

`configure sso`. Para obter mais informações sobre esse comando, consulte Como configurar a CLI da AWS para usar o Centro de identidade do IAM no Guia do usuário do AWS Command Line Interface .

O URL é semelhante ao seguinte:

```
[profile my-sso-profile]
sso_start_url = https://my-sso-portal.awsapps.com/start
sso_region = us-west-2
sso_account_id = 123456789012
sso_role_name = SSOReadOnlyRole
```

As configurações do novo perfil estão definidas abaixo. As duas primeiras configurações definem o portal de AWS acesso. As outras duas configurações são um par que, juntas, definem as permissões que foram configuradas para um usuário. Todas as quatro configurações são obrigatórias.

`sso_start_url`

O URL que aponta para o portal de acesso da AWS da organização. Para encontrar esse valor, abra o console do Centro de identidade do IAM, selecione Configurações e encontre URL do portal.

`sso_region`

O Região da AWS que contém o host do portal de acesso. Essa é a região que foi selecionada quando você ativou o Centro de identidade do IAM. Ela pode ser diferente das regiões que você usa para outras tarefas.

Para obter uma lista completa dos Regiões da AWS e de seus códigos, consulte Endpoints regionais no Referência geral da Amazon Web Services.

`sso_account_id`

O ID de um Conta da AWS que foi adicionado por meio do AWS Organizations serviço. Para ver a lista de contas disponíveis, acesse o console do IAM Identity Center e abra a página de Contas da AWS. O ID da conta que você escolher para essa configuração corresponderá ao valor que você planeja atribuir à configuração do `sso_role_name`, que é mostrado a seguir.

`sso_role_name`

O nome de um conjunto de permissões do Centro de identidade do IAM. Esse conjunto de permissões define as permissões que um usuário recebe por meio do Centro de identidade do IAM.

O procedimento a seguir é uma forma de encontrar o valor dessa configuração.

1. Acesse o console do Centro de Identidade do IAM e habilite a página das Contas da AWS.
2. Escolha uma conta para exibir os detalhes. A conta que você escolher será aquela que contém o usuário ou grupo de SSO ao qual você deseja conceder permissões de SSO.
3. Veja a lista de usuários e grupos atribuídos à conta e encontre o usuário ou grupo de interesse. O conjunto de permissões que você especifica na configuração do `sso_role_name` é um dos conjuntos associados a esse usuário ou grupo.

Ao dar um valor a essa configuração, use o nome do conjunto de permissões, não o nome do recurso da Amazon (ARN).

Os conjuntos de permissões têm políticas do IAM e políticas de permissões personalizadas anexadas a eles. Para obter mais informações, consulte Conjuntos de permissões no Guia do usuário do Centro de Identidade do IAM.

Como gerar e usar de tokens de SSO

Para usar o SSO, o usuário deve primeiro gerar um token temporário e depois usar esse token para acessar AWS aplicativos e recursos apropriados. É possível usar os métodos a seguir para gerar e usar esses tokens temporários:

- Crie aplicativos .NET que gerem primeiro um token, se necessário, e depois use o token.
- Gere um token com o AWS CLI e, em seguida, use o token em aplicativos.NET.

Esses métodos estão descritos nas seções a seguir e demonstrados nos tutoriais.

> **Important**
>
> Seu aplicativo deve fazer referência aos seguintes NuGet pacotes para que a resolução de SSO funcione:
>
> - `AWSSDK.SSO`
> - `AWSSDK.SSOOIDC`
>
> A falha em referenciar esses pacotes resultará em uma exceção de tempo de execução.

Somente aplicativo .NET

Esta seção mostra como criar um aplicativo .NET que gera um token SSO temporário, se necessário, e depois usa esse token. Para obter um tutorial completo desse processo, consulte Tutorial para SSO usando somente aplicativos .NET.

Gere e use um token SSO de forma programática

Além de usar o AWS CLI, você também pode gerar um token SSO programaticamente.

Para fazer isso, seu aplicativo cria um objeto `AWSCredentials` para o perfil SSO, que carrega credenciais temporárias, se houver alguma disponível. Em seguida, seu aplicativo deve converter o objeto `AWSCredentials` em um objeto `SSOAWSCredentials` e definir algumas propriedades de Opções, incluindo um método de retorno de chamada usado para solicitar ao usuário informações de login, se necessário.

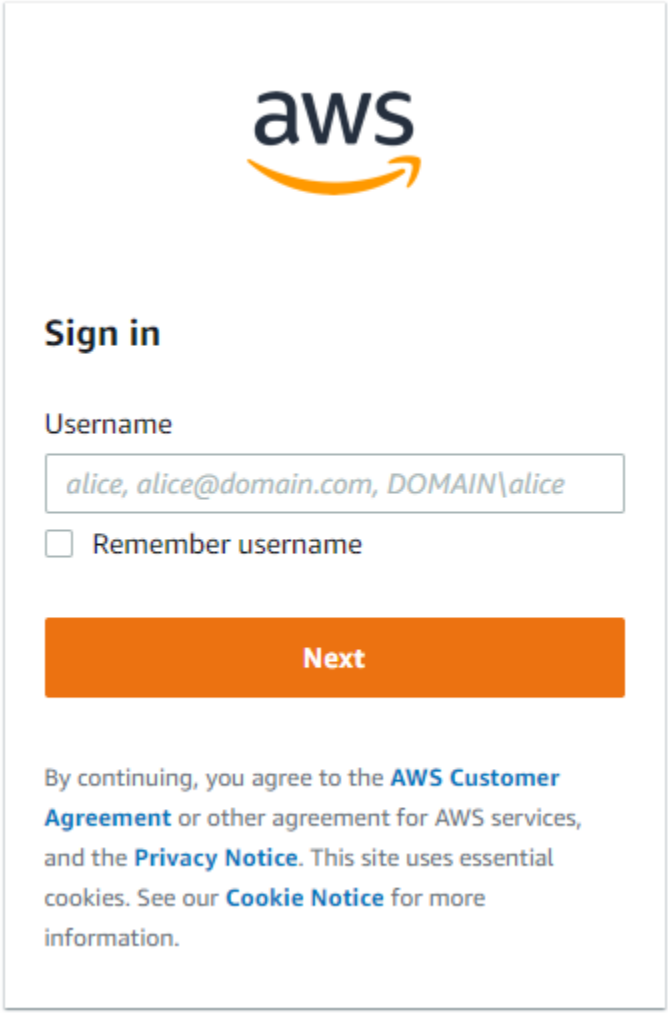
Esse método é mostrado no seguinte trecho de código.

> **Important**
>
> Seu aplicativo deve fazer referência aos seguintes NuGet pacotes para que a resolução de SSO funcione:
>
> - `AWSSDK.SSO`
> - `AWSSDK.SSOOIDC`
>
> A falha em referenciar esses pacotes resultará em uma exceção de tempo de execução.

```
static AWSCredentials LoadSsoCredentials()
{
    var chain = new CredentialProfileStoreChain();
    if (!chain.TryGetAWSCredentials("my-sso-profile", out var credentials))
        throw new Exception("Failed to find the my-sso-profile profile");

    var ssoCredentials = credentials as SSOAWSCredentials;

    ssoCredentials.Options.ClientName = "Example-SSO-App";
    ssoCredentials.Options.SsoVerificationCallback = args =>
    {
```

```
        // Launch a browser window that prompts the SSO user to complete an SSO sign-
in.
        // This method is only invoked if the session doesn't already have a valid SSO
 token.
        // NOTE: Process.Start might not support launching a browser on macOS or Linux.
 If not,
        //       use an appropriate mechanism on those systems instead.
        Process.Start(new ProcessStartInfo
        {
            FileName = args.VerificationUriComplete,
            UseShellExecute = true
        });
    };
    ssoCredentials.Options.SupportsGettingNewToken = true;

    return ssoCredentials;
}
```

Se um token de SSO apropriado não estiver disponível, a janela padrão do navegador será aberta e a página de login apropriada será aberta. Por exemplo, se você estiver usando o Centro de identidade do IAM como fonte de identidade, o usuário verá uma página de login semelhante à seguinte:



> **Note**
>
> A string que você fornece para `SSOAWSCredentials.Options.ClientName` não pode ter espaços. Se a string tiver espaços, você receberá uma exceção de runtime.

Tutorial para SSO usando somente aplicativos .NET

AWS CLI e aplicativo.NET

Esta seção mostra como gerar um token SSO temporário usando o AWS CLI e como usar esse token em um aplicativo. Para obter um tutorial completo desse processo, consulte Tutorial para SSO usando os aplicativos AWS CLI e.NET.

Gere um token de SSO usando o AWS CLI

Além de gerar um token de SSO temporário programaticamente, você usa o AWS CLI para gerar o token. O procedimento a seguir mostra como obter essas informações.

Depois que o usuário cria um perfil habilitado para SSO, conforme mostrado na seção anterior, ele executa o comando `aws sso login` a partir da AWS CLI. Ele deve ter certeza de incluir o parâmetro `--profile` com o nome do perfil habilitado para SSO. Isso é mostrado no exemplo a seguir:

```
aws sso login --profile my-sso-profile
```

Se o usuário quiser gerar um novo token temporário após a expiração do atual, ele poderá executar o mesmo comando novamente.

Use o token SSO gerado em um aplicativo .NET

As informações a seguir mostram como usar um token temporário que já foi gerado.

> **Important**
>
> Seu aplicativo deve fazer referência aos seguintes NuGet pacotes para que a resolução de SSO funcione:
>
> - `AWSSDK.SSO`
> - `AWSSDK.SSOOIDC`
>
> A falha em referenciar esses pacotes resultará em uma exceção de runtime.

Seu aplicativo cria um objeto `AWSCredentials` para o perfil SSO, que carrega as credenciais temporárias geradas anteriormente pela AWS CLI. Isso é semelhante aos métodos mostrados em Acessar credenciais e perfis em um aplicativo e tem o seguinte formato:

```
static AWSCredentials LoadSsoCredentials()
{
    var chain = new CredentialProfileStoreChain();
    if (!chain.TryGetAWSCredentials("my-sso-profile", out var credentials))
        throw new Exception("Failed to find the my-sso-profile profile");
```

```
    return credentials;
}
```

O objeto `AWSCredentials` é então passado para o construtor de um cliente de serviço. Por exemplo:

```
var S3Client_SSO = new AmazonS3Client(LoadSsoCredentials());
```

> **Note**
>
> Não é necessário usar `AWSCredentials` para carregar credenciais temporárias se seu aplicativo tiver sido criado para usar o perfil `[default]` para SSO. Nesse caso, o aplicativo pode criar clientes AWS de serviço sem parâmetros, semelhantes a "`var client = new AmazonS3Client();`".

Tutorial para SSO usando os aplicativos AWS CLI e.NET

Recursos adicionais

Para obter ajuda adicional, consulte os seguintes recursos.

- O que é o Centro de identidade do IAM?
- Configurando o IAM Identity Center AWS CLI para usar
- Usando as credenciais do IAM Identity Center no AWS Toolkit for Visual Studio

Tutoriais

Tópicos

- Tutorial para SSO usando somente aplicativos .NET
- Tutorial para SSO usando os aplicativos AWS CLI e.NET

Tutorial para SSO usando somente aplicativos .NET

Este tutorial mostra como habilitar o SSO para um aplicativo básico e um usuário de SSO de teste. Ele configura o aplicativo para gerar um token SSO temporário de forma programática em vez de usar a AWS CLI.

Este tutorial mostra uma pequena parte da funcionalidade de SSO no AWS SDK para .NET. Para obter detalhes completos sobre o uso do IAM Identity Center com o AWS SDK para .NET, consulte o tópico com informações básicas. Nesse tópico, consulte especialmente a descrição de alto nível desse cenário na subseção chamada Somente aplicativo .NET.

> **Note**
>
> Várias das etapas deste tutorial ajudam você a configurar serviços como AWS Organizations o IAM Identity Center. Se já realizou essa configuração ou se está interessado apenas no código, você pode pular para a seção com o código de exemplo.

Pré-requisitos

- Configure seu ambiente de desenvolvimento, caso ainda não tenha feito isso. Isso é descrito em seções como Instalando e configurando seu conjunto de ferramentas para o AWS SDK para .NET e Conceitos básicos.
- Identifique ou crie pelo menos um Conta da AWS que você possa usar para testar o SSO. Para os fins deste tutorial, isso é chamado de Conta da AWS de teste ou simplesmente conta de teste.
- Identifique um usuário de SSO que possa testar o SSO para você. Essa é uma pessoa que usará o SSO e os aplicativos básicos que você criar. Neste tutorial, essa pessoa pode ser você (o desenvolvedor) ou outra pessoa. Também recomendamos uma configuração na qual o usuário do SSO trabalhe em um computador que não esteja em seu ambiente de desenvolvimento. Contudo, isso não é estritamente necessário.
- O computador do usuário do SSO deve ter uma estrutura do .NET instalada que seja compatível com a que você usou para configurar seu ambiente de desenvolvimento.

Configurar AWS

Esta seção mostra como configurar vários AWS serviços para este tutorial.

Para realizar essa configuração, primeiro entre no teste Conta da AWS como administrador. Faça o seguinte:

Amazon S3

Acesse o console do Amazon S3 e adicione alguns buckets inócuos. Posteriormente neste tutorial, o usuário do SSO recuperará uma lista desses buckets.

AWS IAM

Acesse o console do IAM e adicione alguns usuários do IAM. Se você conceder permissões aos usuários do IAM, limite as permissões a algumas permissões inócuas somente para leitura. Posteriormente neste tutorial, o usuário do SSO recuperará uma lista desses usuários do IAM.

AWS Organizations

Acesse o console do AWS Organizations e ative o Organizations. Para obter mais informações, consulte Criar uma organização no Guia do usuário do AWS Organizations.

Essa ação adiciona o teste Conta da AWS à organização como a conta de gerenciamento. Se você tiver contas de teste adicionais, poderá convidá-las para participar da organização, mas isso não é necessário para este tutorial.

Centro de Identidade do IAM

Acesse o console do Centro de Identidade do IAM e habilite o SSO. Execute a verificação por e-mail, se necessário. Para obter mais informações, consulte Habilitar o Centro de Identidade do IAM no Guia do usuário do Centro de Identidade do IAM.

Depois, execute a configuração a seguir.

Configure o Centro de Identidade do IAM

1. Vá para a página Configurações. Procure a “URL do portal de acesso” e registre o valor para uso posterior na configuração do `sso_start_url`.
2. No banner do Console de gerenciamento da AWS, procure o Região da AWS que foi definido quando você ativou o SSO. Esse é o menu suspenso à esquerda do Conta da AWS ID. Registre o código da região para uso posterior na configuração da `sso_region`. Esse código será semelhante a `us-east-1`.
3. Crie um usuário de SSO da seguinte forma:
    a. Acesse a página Usuários.
    b. Escolha Adicionar usuário e insira o nome de usuário, endereço de e-mail, nome e sobrenome do usuário. Em seguida, escolha Próximo.
    c. Escolha Avançar na página para grupos, revise as informações e escolha Adicionar usuário.
4. Crie um grupo da seguinte forma:

a. Acesse a página Grupos.

b. Escolha Criar grupo e insira o nome do grupo e a descrição do grupo.

c. Na seção Adicionar usuários ao grupo, selecione o usuário de SSO de teste que você criou anteriormente. Selecione Criar grupo.

5. Crie um conjunto de permissões da seguinte forma:

a. Acesse a página Conjuntos de permissões e escolha Criar conjunto de permissões.

b. Em Tipo de conjunto de permissões, selecione Conjunto de permissões personalizado e escolha Avançar.

c. Abra a política em linha e insira a seguinte política:

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "s3:ListAllMyBuckets",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

d. Para este tutorial, insira `SSOReadOnlyRole` como o nome do conjunto de permissões. Adicione uma descrição, se quiser, e escolha Avançar.

e. Revise as informações e escolha Criar.

f. Registre o nome do conjunto de permissões para uso posterior na configuração do `sso_role_name`.

6. Acesse a página de AWS contas e escolha a AWS conta que você adicionou à organização anteriormente.

7. Na seção Visão geral dessa página, encontre o ID da conta e registre-a para uso posterior na configuração do `sso_account_id`.

8. Escolha a guia Usuários e grupos e, em seguida, escolha Atribuir usuários ou grupos.
9. Na página Atribuir usuários e grupos, escolha a guia Grupos, selecione o grupo que você criou anteriormente e escolha Avançar.
10. Selecione o conjunto de permissões que você criou anteriormente e escolha Avançar e, em seguida, escolha Enviar. A configuração leva alguns instantes.

Como criar aplicativos de exemplo

Crie os aplicativos a seguir. Eles serão executados no computador do usuário do SSO.

Listar buckets do Amazon S3

Inclui NuGet pacotes `AWSSDK.SSO` e, `AWSSDK.SSOOIDC` além de `AWSSDK.S3` `AWSSDK.SecurityToken` e.

```
using System;
using System.Threading.Tasks;
using System.Diagnostics;


// NuGet packages: AWSSDK.S3, AWSSDK.SecurityToken, AWSSDK.SSO, AWSSDK.SSOOIDC
using Amazon.Runtime;
using Amazon.Runtime.CredentialManagement;
using Amazon.S3;
using Amazon.S3.Model;
using Amazon.SecurityToken;
using Amazon.SecurityToken.Model;

namespace SSOExample.S3.Programmatic_login
{
    class Program
    {
        // Requirements:
        // - An SSO profile in the SSO user's shared config file.

        // Class members.
        private static string profile = "my-sso-profile";

        static async Task Main(string[] args)
        {
            // Get SSO credentials from the information in the shared config file.
            var ssoCreds = LoadSsoCredentials(profile);
```

```
            // Display the caller's identity.
            var ssoProfileClient = new AmazonSecurityTokenServiceClient(ssoCreds);
            Console.WriteLine($"\nSSO Profile:\n {await
ssoProfileClient.GetCallerIdentityArn()}");

            // Display a list of the account's S3 buckets.
            // The S3 client is created using the SSO credentials obtained earlier.
            var s3Client = new AmazonS3Client(ssoCreds);
            Console.WriteLine("\nGetting a list of your buckets...");
            var listResponse = await s3Client.ListBucketsAsync();
            Console.WriteLine($"Number of buckets: {listResponse.Buckets.Count}");
            foreach (S3Bucket b in listResponse.Buckets)
            {
                Console.WriteLine(b.BucketName);
            }
            Console.WriteLine();
        }

        // Method to get SSO credentials from the information in the shared config
file.
        static AWSCredentials LoadSsoCredentials(string profile)
        {
            var chain = new CredentialProfileStoreChain();
            if (!chain.TryGetAWSCredentials(profile, out var credentials))
                throw new Exception($"Failed to find the {profile} profile");

            var ssoCredentials = credentials as SSOAWSCredentials;

            ssoCredentials.Options.ClientName = "Example-SSO-App";
            ssoCredentials.Options.SsoVerificationCallback = args =>
            {
                // Launch a browser window that prompts the SSO user to complete an SSO
login.
                //  This method is only invoked if the session doesn't already have a
valid SSO token.
                // NOTE: Process.Start might not support launching a browser on macOS
or Linux. If not,
                //       use an appropriate mechanism on those systems instead.
                Process.Start(new ProcessStartInfo
                {
                    FileName = args.VerificationUriComplete,
                    UseShellExecute = true
                });
```

```
            };
            ssoCredentials.Options.SupportsGettingNewToken = true;

            return ssoCredentials;
        }

    }

    // Class to read the caller's identity.
    public static class Extensions
    {
        public static async Task<string> GetCallerIdentityArn(this
 IAmazonSecurityTokenService stsClient)
        {
            var response = await stsClient.GetCallerIdentityAsync(new
 GetCallerIdentityRequest());
            return response.Arn;
        }
    }
}
```

Listar usuários do IAM

Inclui NuGet pacotes `AWSSDK.SSO` e, `AWSSDK.SSOOIDC` além de `AWSSDK.IdentityManagement` `AWSSDK.SecurityToken` e.

```
using System;
using System.Threading.Tasks;
using System.Diagnostics;


// NuGet packages: AWSSDK.IdentityManagement, AWSSDK.SecurityToken, AWSSDK.SSO,
 AWSSDK.SSOOIDC
using Amazon.Runtime;
using Amazon.Runtime.CredentialManagement;
using Amazon.IdentityManagement;
using Amazon.IdentityManagement.Model;
using Amazon.SecurityToken;
using Amazon.SecurityToken.Model;

namespace SSOExample.IAM.Programmatic_login
{
    class Program
```

```
    {
        // Requirements:
        // - An SSO profile in the SSO user's shared config file.

        // Class members.
        private static string profile = "my-sso-profile";

        static async Task Main(string[] args)
        {
            // Get SSO credentials from the information in the shared config file.
            var ssoCreds = LoadSsoCredentials(profile);

            // Display the caller's identity.
            var ssoProfileClient = new AmazonSecurityTokenServiceClient(ssoCreds);
            Console.WriteLine($"\nSSO Profile:\n {await
ssoProfileClient.GetCallerIdentityArn()}");

            // Display a list of the account's IAM users.
            // The IAM client is created using the SSO credentials obtained earlier.
            var iamClient = new AmazonIdentityManagementServiceClient(ssoCreds);
            Console.WriteLine("\nGetting a list of IAM users...");
            var listResponse = await iamClient.ListUsersAsync();
            Console.WriteLine($"Number of IAM users: {listResponse.Users.Count}");
            foreach (User u in listResponse.Users)
            {
                Console.WriteLine(u.UserName);
            }
            Console.WriteLine();
        }

        // Method to get SSO credentials from the information in the shared config
file.
        static AWSCredentials LoadSsoCredentials(string profile)
        {
            var chain = new CredentialProfileStoreChain();
            if (!chain.TryGetAWSCredentials(profile, out var credentials))
                throw new Exception($"Failed to find the {profile} profile");

            var ssoCredentials = credentials as SSOAWSCredentials;

            ssoCredentials.Options.ClientName = "Example-SSO-App";
            ssoCredentials.Options.SsoVerificationCallback = args =>
            {
```

```
                // Launch a browser window that prompts the SSO user to complete an SSO
 login.
                //  This method is only invoked if the session doesn't already have a
 valid SSO token.
                // NOTE: Process.Start might not support launching a browser on macOS
 or Linux. If not,
                //       use an appropriate mechanism on those systems instead.
                Process.Start(new ProcessStartInfo
                {
                    FileName = args.VerificationUriComplete,
                    UseShellExecute = true
                });
            };
            ssoCredentials.Options.SupportsGettingNewToken = true;

            return ssoCredentials;
        }

    }

    // Class to read the caller's identity.
    public static class Extensions
    {
        public static async Task<string> GetCallerIdentityArn(this
 IAmazonSecurityTokenService stsClient)
        {
            var response = await stsClient.GetCallerIdentityAsync(new
 GetCallerIdentityRequest());
            return response.Arn;
        }
    }
}
```

Além de exibir listas de buckets do Amazon S3 e usuários do IAM, esses aplicativos exibem o ARN de identidade do usuário para o perfil habilitado para SSO, que está neste tutorial. `my-sso-profile`

Esses aplicativos realizam tarefas de login com SSO fornecendo um método de retorno de chamada na propriedade Options de um objeto. SSOAWSCredentials

Instrua o usuário do SSO

Peça ao usuário do SSO que verifique seu e-mail e aceite o convite do SSO. Será solicitado que ele defina uma senha. A mensagem pode levar alguns minutos para chegar à caixa de entrada do usuário do SSO.

Forneça ao usuário do SSO os aplicativos criados anteriormente.

Em seguida, peça ao usuário do SSO que faça o seguinte:

1. Se a pasta que contém o AWS `config` arquivo compartilhado não existir, crie-a. Se a pasta existir e tiver uma subpasta chamada `.sso`, exclua essa subpasta.

   A localização dessa pasta geralmente é `%USERPROFILE%\.aws` no Windows e `~/.aws` no Linux e no macOS.
2. Crie um AWS `config` arquivo compartilhado nessa pasta, se necessário, e adicione um perfil a ele da seguinte maneira:

   ```
   [default]
   region = <default Region>

   [profile my-sso-profile]
   sso_start_url = <user portal URL recorded earlier>
   sso_region = <Region code recorded earlier>
   sso_account_id = <account ID recorded earlier>
   sso_role_name = SSOReadOnlyRole
   ```
3. Execute o aplicativo do Amazon S3.
4. Na página de login na web resultante, faça login. Use o nome de usuário da mensagem de convite e a senha que foi criada em resposta à mensagem.
5. Quando o login estiver concluído, o aplicativo exibirá a lista de buckets do S3.
6. Execute o aplicativo do IAM. O aplicativo exibe a lista de usuários do IAM. Isso é verdade mesmo que um segundo login não tenha sido realizado. O aplicativo IAM usa o token temporário que foi criado anteriormente.

Limpeza

Se você não quiser manter os recursos que criou durante este tutorial, apague-os. Eles podem ser AWS recursos ou recursos em seu ambiente de desenvolvimento, como arquivos e pastas.

Tutorial para SSO usando os aplicativos AWS CLI e.NET

Este tutorial mostra como habilitar o SSO para um aplicativo .NET básico e um usuário de SSO de teste. Ele usa o AWS CLI para gerar um token SSO temporário em vez de gerá-lo programaticamente.

Este tutorial mostra uma pequena parte da funcionalidade de SSO no AWS SDK para .NET. Para obter detalhes completos sobre o uso do IAM Identity Center com o AWS SDK para .NET, consulte o tópico com informações básicas. Nesse tópico, consulte especialmente a descrição de alto nível desse cenário na subseção chamada AWS CLI e aplicativo.NET.

> **Note**
>
> Várias das etapas deste tutorial ajudam você a configurar serviços como AWS Organizations o IAM Identity Center. Se já realizou essas configurações ou se está interessado apenas no código, você pode pular para a seção com o código de exemplo.

Pré-requisitos

- Configure seu ambiente de desenvolvimento, caso ainda não tenha feito isso. Isso é descrito em seções como Instalando e configurando seu conjunto de ferramentas para o AWS SDK para .NET e Conceitos básicos.
- Identifique ou crie pelo menos um Conta da AWS que você possa usar para testar o SSO. Para os fins deste tutorial, isso é chamado de Conta da AWS de teste ou simplesmente conta de teste.
- Identifique um usuário de SSO que possa testar o SSO para você. Essa é uma pessoa que usará o SSO e os aplicativos básicos que você criar. Neste tutorial, essa pessoa pode ser você (o desenvolvedor) ou outra pessoa. Também recomendamos uma configuração na qual o usuário do SSO trabalhe em um computador que não esteja em seu ambiente de desenvolvimento. Contudo, isso não é estritamente necessário.
- O computador do usuário do SSO deve ter uma estrutura do .NET instalada que seja compatível com a que você usou para configurar seu ambiente de desenvolvimento.
- Certifique-se de que a AWS CLI versão 2 esteja instalada no computador do usuário do SSO. Você pode verificar isso executando `aws --version` em um prompt de comando ou terminal.

Configurar AWS

Esta seção mostra como configurar vários AWS serviços para este tutorial.

Para realizar essa configuração, primeiro entre no teste Conta da AWS como administrador. Faça o seguinte:

Amazon S3

Acesse o console do Amazon S3 e adicione alguns buckets inócuos. Posteriormente neste tutorial, o usuário do SSO recuperará uma lista desses buckets.

AWS IAM

Acesse o console do IAM e adicione alguns usuários do IAM. Se você conceder permissões aos usuários do IAM, limite as permissões a algumas permissões inócuas somente para leitura. Posteriormente neste tutorial, o usuário do SSO recuperará uma lista desses usuários do IAM.

AWS Organizations

Acesse o console do AWS Organizations e ative o Organizations. Para obter mais informações, consulte Criar uma organização no Guia do usuário do AWS Organizations.

Essa ação adiciona o teste Conta da AWS à organização como a conta de gerenciamento. Se você tiver contas de teste adicionais, poderá convidá-las para participar da organização, mas isso não é necessário para este tutorial.

Centro de Identidade do IAM

Acesse o console do Centro de Identidade do IAM e habilite o SSO. Execute a verificação por e-mail, se necessário. Para obter mais informações, consulte Habilitar o Centro de Identidade do IAM no Guia do usuário do Centro de Identidade do IAM.

Depois, execute a configuração a seguir.

Configure o Centro de Identidade do IAM

1. Vá para a página Configurações. Procure a “URL do portal de acesso” e registre o valor para uso posterior na configuração do `sso_start_url`.
2. No banner do Console de gerenciamento da AWS, procure o Região da AWS que foi definido quando você ativou o SSO. Esse é o menu suspenso à esquerda do Conta da AWS ID. Registre

o código da região para uso posterior na configuração da `sso_region`. Esse código será semelhante a `us-east-1`.

3. Crie um usuário de SSO da seguinte forma:

   a. Acesse a página Usuários.

   b. Escolha Adicionar usuário e insira o nome de usuário, endereço de e-mail, nome e sobrenome do usuário. Em seguida, escolha Próximo.

   c. Escolha Avançar na página para grupos, revise as informações e escolha Adicionar usuário.

4. Crie um grupo da seguinte forma:

   a. Acesse a página Grupos.

   b. Escolha Criar grupo e insira o nome do grupo e a descrição do grupo.

   c. Na seção Adicionar usuários ao grupo, selecione o usuário de SSO de teste que você criou anteriormente. Selecione Criar grupo.

5. Crie um conjunto de permissões da seguinte forma:

   a. Acesse a página Conjuntos de permissões e escolha Criar conjunto de permissões.

   b. Em Tipo de conjunto de permissões, selecione Conjunto de permissões personalizado e escolha Avançar.

   c. Abra a política em linha e insira a seguinte política:

      JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "s3:ListAllMyBuckets",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

   d. Para este tutorial, insira `SSOReadOnlyRole` como o nome do conjunto de permissões. Adicione uma descrição, se quiser, e escolha Avançar.
   e. Revise as informações e escolha Criar.
   f. Registre o nome do conjunto de permissões para uso posterior na configuração do `sso_role_name`.
6. Acesse a página de AWS contas e escolha a AWS conta que você adicionou à organização anteriormente.
7. Na seção Visão geral dessa página, encontre o ID da conta e registre-a para uso posterior na configuração do `sso_account_id`.
8. Escolha a guia Usuários e grupos e, em seguida, escolha Atribuir usuários ou grupos.
9. Na página Atribuir usuários e grupos, escolha a guia Grupos, selecione o grupo que você criou anteriormente e escolha Avançar.
10. Selecione o conjunto de permissões que você criou anteriormente e escolha Avançar e, em seguida, escolha Enviar. A configuração leva alguns instantes.

Como criar aplicativos de exemplo

Crie os aplicativos a seguir. Eles serão executados no computador do usuário do SSO.

Listar buckets do Amazon S3

Inclui NuGet pacotes `AWSSDK.SSO` e, `AWSSDK.SSOOIDC` além de `AWSSDK.S3` `AWSSDK.SecurityToken` e.

```
using System;
using System.Threading.Tasks;

// NuGet packages: AWSSDK.S3, AWSSDK.SecurityToken, AWSSDK.SSO, AWSSDK.SSOOIDC
using Amazon.Runtime;
using Amazon.Runtime.CredentialManagement;
using Amazon.S3;
using Amazon.S3.Model;
using Amazon.SecurityToken;
using Amazon.SecurityToken.Model;

namespace SSOExample.S3.CLI_login
{
    class Program
```

```
    {
        // Requirements:
        // - An SSO profile in the SSO user's shared config file.
        // - An active SSO Token.
        //    If an active SSO token isn't available, the SSO user should do the
 following:
        //    In a terminal, the SSO user must call "aws sso login --profile my-sso-
profile".

        // Class members.
        private static string profile = "my-sso-profile";
        static async Task Main(string[] args)
        {
            // Get SSO credentials from the information in the shared config file.
            var ssoCreds = LoadSsoCredentials(profile);

            // Display the caller's identity.
            var ssoProfileClient = new AmazonSecurityTokenServiceClient(ssoCreds);
            Console.WriteLine($"\nSSO Profile:\n {await
 ssoProfileClient.GetCallerIdentityArn()}");

            // Display a list of the account's S3 buckets.
            // The S3 client is created using the SSO credentials obtained earlier.
            var s3Client = new AmazonS3Client(ssoCreds);
            Console.WriteLine("\nGetting a list of your buckets...");
            var listResponse = await s3Client.ListBucketsAsync();
            Console.WriteLine($"Number of buckets: {listResponse.Buckets.Count}");
            foreach (S3Bucket b in listResponse.Buckets)
            {
                Console.WriteLine(b.BucketName);
            }
            Console.WriteLine();
        }

        // Method to get SSO credentials from the information in the shared config
 file.
        static AWSCredentials LoadSsoCredentials(string profile)
        {
            var chain = new CredentialProfileStoreChain();
            if (!chain.TryGetAWSCredentials(profile, out var credentials))
                throw new Exception($"Failed to find the {profile} profile");
            return credentials;
        }
    }
```

```
    // Class to read the caller's identity.
    public static class Extensions
    {
        public static async Task<string> GetCallerIdentityArn(this
 IAmazonSecurityTokenService stsClient)
        {
            var response = await stsClient.GetCallerIdentityAsync(new
 GetCallerIdentityRequest());
            return response.Arn;
        }
    }
}
```

Listar usuários do IAM

Inclui NuGet pacotes `AWSSDK.SSO` e, `AWSSDK.SSOOIDC` além de `AWSSDK.IdentityManagement` `AWSSDK.SecurityToken` e.

```
using System;
using System.Threading.Tasks;

// NuGet packages: AWSSDK.IdentityManagement, AWSSDK.SecurityToken, AWSSDK.SSO,
 AWSSDK.SSOOIDC
using Amazon.Runtime;
using Amazon.Runtime.CredentialManagement;
using Amazon.IdentityManagement;
using Amazon.IdentityManagement.Model;
using Amazon.SecurityToken;
using Amazon.SecurityToken.Model;

namespace SSOExample.IAM.CLI_login
{
    class Program
    {
        // Requirements:
        // - An SSO profile in the SSO user's shared config file.
        // - An active SSO Token.
        //    If an active SSO token isn't available, the SSO user should do the
 following:
        //    In a terminal, the SSO user must call "aws sso login --profile my-sso-
profile".
```

```
        // Class members.
        private static string profile = "my-sso-profile";
        static async Task Main(string[] args)
        {
            // Get SSO credentials from the information in the shared config file.
            var ssoCreds = LoadSsoCredentials(profile);

            // Display the caller's identity.
            var ssoProfileClient = new AmazonSecurityTokenServiceClient(ssoCreds);
            Console.WriteLine($"\nSSO Profile:\n {await
ssoProfileClient.GetCallerIdentityArn()}");

            // Display a list of the account's IAM users.
            // The IAM client is created using the SSO credentials obtained earlier.
            var iamClient = new AmazonIdentityManagementServiceClient(ssoCreds);
            Console.WriteLine("\nGetting a list of IAM users...");
            var listResponse = await iamClient.ListUsersAsync();
            Console.WriteLine($"Number of IAM users: {listResponse.Users.Count}");
            foreach (User u in listResponse.Users)
            {
                Console.WriteLine(u.UserName);
            }
            Console.WriteLine();
        }

        // Method to get SSO credentials from the information in the shared config
file.
        static AWSCredentials LoadSsoCredentials(string profile)
        {
            var chain = new CredentialProfileStoreChain();
            if (!chain.TryGetAWSCredentials(profile, out var credentials))
                throw new Exception($"Failed to find the {profile} profile");
            return credentials;
        }
    }

    // Class to read the caller's identity.
    public static class Extensions
    {
        public static async Task<string> GetCallerIdentityArn(this
IAmazonSecurityTokenService stsClient)
        {
            var response = await stsClient.GetCallerIdentityAsync(new
GetCallerIdentityRequest());
```

```
            return response.Arn;
        }
    }
}
```

Além de exibir listas de buckets do Amazon S3 e usuários do IAM, esses aplicativos exibem o ARN de identidade do usuário para o perfil habilitado para SSO, que está neste tutorial. `my-sso-profile`

Instrua o usuário do SSO

Peça ao usuário do SSO que verifique seu e-mail e aceite o convite do SSO. Será solicitado que ele defina uma senha. A mensagem pode levar alguns minutos para chegar à caixa de entrada do usuário do SSO.

Forneça ao usuário do SSO os aplicativos criados anteriormente.

Em seguida, peça ao usuário do SSO que faça o seguinte:

1. Se a pasta que contém o AWS `config` arquivo compartilhado não existir, crie-a. Se a pasta existir e tiver uma subpasta chamada `.sso`, exclua essa subpasta.

   A localização dessa pasta geralmente é `%USERPROFILE%\.aws` no Windows e `~/.aws` no Linux e no macOS.
2. Crie um AWS `config` arquivo compartilhado nessa pasta, se necessário, e adicione um perfil a ele da seguinte maneira:

   ```
   [default]
   region = <default Region>

   [profile my-sso-profile]
   sso_start_url = <user portal URL recorded earlier>
   sso_region = <Region code recorded earlier>
   sso_account_id = <account ID recorded earlier>
   sso_role_name = SSOReadOnlyRole
   ```
3. Execute o aplicativo do Amazon S3. Uma exceção de tempo de execução é exibida.
4. Execute o seguinte AWS CLI comando:

   ```
   aws sso login --profile my-sso-profile
   ```

5. Na página de login na web resultante, faça login. Use o nome de usuário da mensagem de convite e a senha que foi criada em resposta à mensagem.
6. Execute o aplicativo do Amazon S3 novamente. Agora, o aplicativo exibe a lista de buckets do S3.
7. Execute o aplicativo do IAM. O aplicativo exibe a lista de usuários do IAM. Isso é verdade mesmo que um segundo login não tenha sido realizado. O aplicativo IAM usa o token temporário que foi criado anteriormente.

Limpeza

Se você não quiser manter os recursos que criou durante este tutorial, apague-os. Eles podem ser AWS recursos ou recursos em seu ambiente de desenvolvimento, como arquivos e pastas.

Implemente aplicativos em AWS

Depois de desenvolver seu aplicativo ou serviço do. NET Core nativo de nuvem em uma máquina de desenvolvimento, você vai desejar implantá-lo na AWS. Você pode fazer isso usando o Console de gerenciamento da AWS ou determinados serviços, como CloudFormation ou AWS Cloud Development Kit (AWS CDK). Você também pode usar AWS ferramentas que foram criadas para fins de implantação. Usando essas ferramentas, você pode fazer o seguinte:

Implantar a partir da CLI do .NET

Você pode usar as seguintes AWS ferramentas para .NET CLI para implantar seus aplicativos em: AWS

- AWS Ferramenta de implantação para .NET CLI - Suporta implantações no AWS App RunnerAmazon Elastic Container Service (Amazon ECS) e. AWS Elastic Beanstalk
- AWS Lambda Ferramentas para .NET CLI - Suporta implantações de projetos. AWS Lambda

Implante a partir dos kits de ferramentas IDE

Você pode usar AWS kits de ferramentas para implantar seus aplicativos diretamente do IDE de sua escolha:

- AWS Toolkit for Visual Studio

  > **Note**
  >
  > O recurso “Publicar em AWS” no kit de ferramentas expõe a mesma funcionalidade da Ferramenta de AWS Implantação para CLI do.NET. Para saber mais, acesse Publicar na AWS no Guia do usuário do AWS Toolkit for Visual Studio .

- AWS Toolkit for JetBrains

  Consulte Trabalhar com aplicativos AWS sem servidor e Trabalhar com. AWS App Runner

- Toolkit for VS Code da AWS

  Consulte Como trabalhar com aplicativos sem servidor e Como usar AWS App Runner.

- Kit de ferramentas da AWS para Azure DevOps

Casos de uso

As seções a seguir contêm cenários de casos de uso para determinados tipos de aplicativos, incluindo informações sobre como você usaria a CLI do.NET para implantar esses aplicativos.

- Aplicativos ASP.NET Core:
- Aplicativos do .NET Console
- Aplicativos Blazor WebAssembly
- AWS Lambda projetos

Aplicativos ASP.NET Core:

A ferramenta de implantação da AWS para a CLI do .NET o ajuda a implantar seus aplicativos do ASP.NET e serve como orientação em um processo de implantação. É uma ferramenta interativa para a CLI do.NET que ajuda a implantar aplicativos .NET com conhecimento mínimo sobra a AWS .

A ferramenta de implantação tem os seguintes recursos:

- Recomendações de computação para seu aplicativo - Obtenha as recomendações de computação e saiba qual AWS computação é mais adequada para seu aplicativo.
- Geração de Dockerfile: a ferramenta gera um Dockerfile, se necessário, ou usa um Dockerfile existente.
- Empacotamento e implantação automáticos — A ferramenta cria os artefatos de implantação, provisiona a infraestrutura usando um projeto de AWS CDK implantação gerado e implanta seu aplicativo na computação escolhida. AWS
- Implantações reproduzíveis e compartilháveis — Você pode gerar e modificar projetos de AWS CDK implantação de acordo com seu caso de uso específico. Você também pode controlar a versão de seus projetos e compartilhá-los com sua equipe para implantações reproduzíveis.
- Ajuda no aprendizado AWS CDK do.NET - A ferramenta ajuda você a aprender gradualmente AWS as ferramentas subjacentes nas quais ela se baseia, como AWS CDK o.

A Ferramenta de AWS Implantação oferece suporte à implantação de aplicativos ASP.NET Core nos seguintes serviços: AWS

- Uso do Amazon ECS Service AWS Fargate- Suporta implantações de aplicativos web no Amazon Elastic Container Service (Amazon ECS) com poder computacional gerenciado por um mecanismo de computação sem servidor. AWS Fargate
- AWS App Runner- Oferece suporte a implantações em um serviço totalmente gerenciado que facilita aos desenvolvedores a implantação de aplicativos web em contêineres e APIs em grande escala. Não é necessária nenhuma experiência prévia em infraestrutura.
- AWS Elastic Beanstalk- Oferece suporte a implantações em um serviço que facilita para os desenvolvedores a implantação de aplicativos web e APIs em um ambiente totalmente gerenciado em grande escala. Não é necessária nenhuma experiência prévia em infraestrutura.

Para saber mais, consulte a visão geral da ferramenta. Para começar a partir dali, navegue até Documentação, Conceitos Básicos e escolha Como instalar para obter instruções de instalação.

Aplicativos do .NET Console

A ferramenta de implantação da AWS para a CLI do.NET ajuda você a implantar seus aplicativos do.NET Console como um serviço ou uma tarefa agendada como uma imagem de contêiner no Linux e serve como orientação em um processo de implantação. Se seu aplicativo não tiver um Dockerfile, a ferramenta o gerará automaticamente. Caso contrário, um Dockerfile existente será usado.

A ferramenta de implantação tem os seguintes recursos:

- Recomendações de computação para seu aplicativo - Obtenha as recomendações de computação e saiba qual AWS computação é mais adequada para seu aplicativo.
- Geração de Dockerfile: a ferramenta gera um Dockerfile, se necessário, ou usa um Dockerfile existente.
- Empacotamento e implantação automáticos — A ferramenta cria os artefatos de implantação, provisiona a infraestrutura usando um projeto de AWS CDK implantação gerado e implanta seu aplicativo na computação escolhida. AWS
- Implantações reproduzíveis e compartilháveis — Você pode gerar e modificar projetos de AWS CDK implantação de acordo com seu caso de uso específico. Você também pode controlar a versão de seus projetos e compartilhá-los com sua equipe para implantações reproduzíveis.
- Ajuda no aprendizado AWS CDK do.NET - A ferramenta ajuda você a aprender gradualmente AWS as ferramentas subjacentes nas quais ela se baseia, como AWS CDK o.

A Ferramenta de AWS Implantação oferece suporte à implantação de aplicativos do.NET Console.NET nos seguintes AWS serviços:

- Uso do Amazon ECS Service AWS Fargate- Suporta implantações de aplicativos.NET como um serviço (por exemplo, um processador em segundo plano) no Amazon Elastic Container Service (Amazon ECS) com poder computacional gerenciado por um mecanismo de computação sem servidor. AWS Fargate
- Uso de tarefas agendadas do Amazon ECS AWS Fargate- Suporta implantações de aplicativos.NET como uma tarefa programada (por exemplo, end-of-day processo) no Amazon ECS com poder computacional gerenciado por AWS Fargate um mecanismo de computação sem servidor.

Para saber mais, consulte a visão geral da ferramenta. Para começar a partir dali, navegue até Documentação, Conceitos Básicos e escolha Como instalar para obter instruções de instalação.

Aplicativos Blazor WebAssembly

A ferramenta de AWS implantação para a CLI do.NET ajuda você a hospedar seu aplicativo Blazor no WebAssembly Amazon S3, usando a Amazon para entrega na rede de conteúdo. CloudFront Seu aplicativo é implantado em um bucket do S3 para hospedagem na web. A ferramenta cria e configura um bucket S3 e, em seguida, carrega seu aplicativo Blazor no bucket.

A ferramenta de implantação tem os seguintes recursos:

- Empacotamento e implantação automáticos — A ferramenta cria os artefatos de implantação, provisiona a infraestrutura usando um projeto de AWS CDK implantação gerado e implanta seu aplicativo na computação escolhida. AWS
- Implantações reproduzíveis e compartilháveis — Você pode gerar e modificar projetos de AWS CDK implantação de acordo com seu caso de uso específico. Você também pode controlar a versão de seus projetos e compartilhá-los com sua equipe para implantações reproduzíveis.
- Ajuda no aprendizado AWS CDK do.NET - A ferramenta ajuda você a aprender gradualmente AWS as ferramentas subjacentes nas quais ela se baseia, como AWS CDK o.

Para saber mais, consulte a visão geral da ferramenta. Para começar a partir dali, navegue até Documentação, Conceitos Básicos e escolha Como instalar para obter instruções de instalação.

AWS Lambda projetos

AWS Lambda é um serviço de computação que permite executar código sem provisionar ou gerenciar servidores. Ele executa seu código em uma infraestrutura de computação de alta disponibilidade e executa toda a administração dos recursos computacionais. Para obter mais informações sobre o Lambda, consulte O que é o AWS Lambda? no Guia do AWS Lambda desenvolvedor.

Você pode implantar funções do Lambda usando a interface da linha de comandos (CLI) do .NET.

Tópicos

- Pré-requisitos
- Comandos do Lambda disponíveis
- Etapas para implantar

Pré-requisitos

Antes de começar a usar a CLI do .NET para implantar funções do Lambda, você deve atender os seguintes pré-requisitos:

- Confirme se você tem a CLI do .NET instalada. Por exemplo: `dotnet --version`. Se necessário, acesse https://dotnet.microsoft.com/download para instalá-lo.
- Configure a CLI do .NET para trabalhar com o Lambda. Para obter uma descrição de como fazer isso, consulte CLI do .NET Core no Guia do Desenvolvedor do AWS Lambda . Nesse procedimento, o seguinte é o comando de implantação:

  ```
  dotnet lambda deploy-function MyFunction --function-role role
  ```

  Se você não tiver certeza de como criar um perfil do IAM para este exercício, não inclua a parte `--function-role role`. A ferramenta ajudará você a criar um novo perfil.

Comandos do Lambda disponíveis

Para listar os comandos do Lambda que estão disponíveis por meio da CLI do .NET, abra um prompt de comando ou terminal e digite `dotnet lambda --help`. A saída desse comando será semelhante a:

```
Amazon Lambda Tools for .NET applications
Project Home: https://github.com/aws/aws-extensions-for-dotnet-cli, https://github.com/
aws/aws-lambda-dotnet

Commands to deploy and manage AWS Lambda functions:

        deploy-function         Command to deploy the project to AWS Lambda
        ...
        (etc.)

To get help on individual commands execute:
        dotnet lambda help <command>
```

A saída lista todos os comandos que estão disponíveis atualmente.

Etapas para implantar

As instruções a seguir pressupõem que você tenha criado um AWS Lambda projeto.NET. Para fins desse procedimento, o projeto é denominado `DotNetCoreLambdaTest`.

1. Abra um prompt de comando ou terminal e navegue até a pasta que contém o arquivo de projeto .NET Lambda.
2. Insira `dotnet lambda deploy-function`.
3. Se solicitado, insira a AWS Região (a região na qual sua função Lambda será implantada).
4. Quando solicitado, digite o nome da função a ser implantada, por exemplo, `DotNetCoreLambdaTest`. Pode ser o nome de uma função que já exista na sua Conta da AWS ou de uma que ainda não tenha sido implantada ali.
5. Quando solicitado, selecione ou crie o perfil do IAM que o Lambda vai pressupor ao executar a função.

Após a conclusão ter sido bem-sucedida, a mensagem Nova função do Lambda criada é exibida.

```
Executing publish command
...
(etc.)
New Lambda function created
```

Se você implantar uma função que já existe em sua conta, a função de implantação solicitará somente a Região da AWS (se necessário). Nesse caso, a saída do comando termina com `Updating code for existing function`.

Depois que a função do Lambda for implantada, ela estará pronta para ser usada. Para obter mais informações, consulte exemplos de como usar o AWS Lambda.

O Lambda monitora automaticamente as funções do Lambda para você e relata métricas por meio da Amazon. CloudWatch Para monitorar e solucionar os problemas da sua função do Lambda, consulte Monitoramento e solução de problemas de aplicativos Lambda.

Ligando para AWS serviços do AWS SDK para .NET

As seções a seguir contêm exemplos, tutoriais, tarefas e guias que mostram como usar o AWS SDK para .NET para trabalhar com AWS serviços. Esses exemplos e tutoriais dependem de uma API que o AWS SDK para .NET fornece. Para ver quais classes e métodos estão disponíveis na API, consulte a Referência de API do AWS SDK para .NET.

Se você é novo no AWS SDK para .NET, talvez queira conferir o Criando um aplicativo simples tópico primeiro. Ele fornece uma introdução ao SDK.

Você pode encontrar mais exemplos de código no Repositório de exemplos de AWS código e no repositório awslabs em. GitHub

Antes de começar, verifique se você configurou seu ambiente e configurou seu projeto. Revise também as informações em Uso da SDK.

Tópicos

- Exemplos de código guiado para o AWS SDK para .NET
- Usando AWS Lambda para serviço de computação
- Bibliotecas e estruturas de alto nível para o AWS SDK para .NET
- Support para outros AWS serviços e configuração

Exemplos de código guiado para o AWS SDK para .NET

As seções a seguir contêm exemplos de código e fornecem orientação para os exemplos. Eles podem ajudá-lo a aprender como usar o AWS SDK para .NET para trabalhar com AWS serviços. Para obter exemplos de código adicionais, consulte Exemplos de código.

> **Note**
>
> Exemplos de código específicos para a V3 do também AWS SDK para .NET estão disponíveis. Para encontrá-los, consulte Exemplos de código com orientação e exemplos de código no Guia do desenvolvedor AWS SDK para .NET (V3). No entanto, lembre-se de que, se você usar os exemplos de código específicos da V3 com a V4 do SDK (a versão mais recente), talvez seja necessário fazer ajustes de acordo com as informações contidas. Migrando para a versão 4

Se você é novo no AWS SDK para .NET, talvez queira conferir o Criando um aplicativo simples tópico primeiro. Ele fornece uma introdução ao SDK.

Antes de começar, verifique se você configurou seu ambiente e configurou seu projeto. Revise também as informações em Uso da SDK.

Tópicos

- Acessando CloudFormation com o AWS SDK para .NET
- Autenticar usuários com o Amazon Cognito
- Como usar os bancos de dados NoSQL Amazon DynamoDB
- Trabalhar com o Amazon EC2
- Acessando AWS Identity and Access Management (IAM) com o AWS SDK para .NET
- Usar o armazenamento para internet Amazon Simple Storage Service
- Enviar notificações da nuvem usando o Amazon Simple Notification Service
- Enviar mensagens usando o Amazon SQS

Acessando CloudFormation com o AWS SDK para .NET

Os AWS SDK para .NET suportes AWS CloudFormation, que criam e provisionam implantações de AWS infraestrutura de forma previsível e repetida.

APIs

Eles AWS SDK para .NET fornecem APIs para CloudFormation clientes. Eles APIs permitem que você trabalhe com CloudFormation recursos como modelos e pilhas. Esta seção contém um pequeno número de exemplos que mostram os padrões que você pode seguir ao trabalhar com eles APIs. Para ver o conjunto completo de APIs, consulte a Referência da AWS SDK para .NET API (e vá até “Amazon. CloudFormation“).

Eles AWS CloudFormation APIs são fornecidos pelo AWSSDK. CloudFormationpacote.

Pré-requisitos

Antes de começar, verifique se você configurou seu ambiente e configurou seu projeto. Revise também as informações em Uso da SDK.

Tópicos

Tópicos

- Listando AWS recursos usando AWS CloudFormation

Listando AWS recursos usando AWS CloudFormation

Este exemplo mostra como usar o AWS SDK para .NET para listar os recursos em CloudFormation pilhas. O exemplo usa a API de baixo nível. O aplicativo não aceita argumentos, apenas reúne informações de todas as pilhas acessíveis às credenciais do usuário e, em seguida, exibe informações sobre essas pilhas.

Referências do SDK

NuGet pacotes:

- AWSSDK.CloudFormation

Elementos de programação:

- Namespace Amazon. CloudFormation

  Classe AmazonCloudFormationClient
- Namespace Amazon. CloudFormation.Modelo

  Classe ICloudFormationPaginatorFactory. DescribeStacks

  Classe DescribeStackResourcesRequest

  Classe DescribeStackResourcesResponse

  Classe Stack

  Classe StackResource

  Classe Tag

```
using Amazon.CloudFormation;
using Amazon.CloudFormation.Model;
using Amazon.Runtime;
```

```
namespace CloudFormationActions;

public static class HelloCloudFormation
{
    public static IAmazonCloudFormation _amazonCloudFormation;

    static async Task Main(string[] args)
    {
        // Create the CloudFormation client
        _amazonCloudFormation = new AmazonCloudFormationClient();
        Console.WriteLine($"\nIn Region:
 {_amazonCloudFormation.Config.RegionEndpoint}");

        // List the resources for each stack
        await ListResources();
    }

    /// <summary>
    /// Method to list stack resources and other information.
    /// </summary>
    /// <returns>True if successful.</returns>
    public static async Task<bool> ListResources()
    {
        try
        {
            Console.WriteLine("Getting CloudFormation stack information...");

            // Get all stacks using the stack paginator.
            var paginatorForDescribeStacks =
                _amazonCloudFormation.Paginators.DescribeStacks(
                    new DescribeStacksRequest());
            await foreach (Stack stack in paginatorForDescribeStacks.Stacks)
            {
                // Basic information for each stack

 Console.WriteLine("\n-----------------------------------------------");
                Console.WriteLine($"\nStack: {stack.StackName}");
                Console.WriteLine($"  Status: {stack.StackStatus.Value}");
                Console.WriteLine($"  Created: {stack.CreationTime}");

                // The tags of each stack (etc.)
                if (stack.Tags.Count > 0)
                {
```

```
                    Console.WriteLine("  Tags:");
                    foreach (Tag tag in stack.Tags)
                        Console.WriteLine($"    {tag.Key}, {tag.Value}");
                }

                // The resources of each stack
                DescribeStackResourcesResponse responseDescribeResources =
                    await _amazonCloudFormation.DescribeStackResourcesAsync(
                        new DescribeStackResourcesRequest
                        {
                            StackName = stack.StackName
                        });
                if (responseDescribeResources.StackResources.Count > 0)
                {
                    Console.WriteLine("  Resources:");
                    foreach (StackResource resource in responseDescribeResources
                                .StackResources)
                        Console.WriteLine(
                            $"    {resource.LogicalResourceId}:
{resource.ResourceStatus}");
                }
            }

            Console.WriteLine("\n---------------------------------------------");
            return true;
        }
        catch (AmazonCloudFormationException ex)
        {
            Console.WriteLine("Unable to get stack information:\n" + ex.Message);
            return false;
        }
        catch (AmazonServiceException ex)
        {
            if (ex.Message.Contains("Unable to get IAM security credentials"))
            {
                Console.WriteLine(ex.Message);
                Console.WriteLine("If you are usnig SSO, be sure to install" +
                                  " the AWSSDK.SSO and AWSSDK.SSOOIDC packages.");
            }
            else
            {
                Console.WriteLine(ex.Message);
                Console.WriteLine(ex.StackTrace);
            }
```

```
            return false;
        }
        catch (ArgumentNullException ex)
        {
            if (ex.Message.Contains("Options property cannot be empty: ClientName"))
            {
                Console.WriteLine(ex.Message);
                Console.WriteLine("If you are using SSO, have you logged in?");
            }
            else
            {
                Console.WriteLine(ex.Message);
                Console.WriteLine(ex.StackTrace);
            }

            return false;
        }
    }
```

Autenticar usuários com o Amazon Cognito

> **Note**
>
> As informações neste tópico são específicas para projetos baseados no.NET Framework e na AWS SDK para .NET versão 3.3 e anteriores.

Usando o Amazon Cognito Identity, você pode criar identidades exclusivas para seus usuários e autenticá-los para acesso seguro aos seus AWS recursos, como o Amazon S3 ou o Amazon DynamoDB. O Amazon Cognito Identity é compatível com provedores de identidade públicos, como o Amazon, o Facebook e o Google, ou qualquer provedor compatível com o OpenID Connect, bem como identidades não autenticadas. O Amazon Cognito é compatível também com as identidades autenticadas pelos desenvolvedores, que permitem registrar e autenticar usuários por meio do processo de autenticação de back-end, sem deixar de usar a Sincronização do Amazon Cognito para sincronizar os dados do usuário e acessar os recursos AWS .

Para obter mais informações sobre o Amazon Cognito, consulte o Amazon Cognito Developer Guide.

Os exemplos de código a seguir mostram como usar facilmente o Amazon Cognito Identity. O exemplo Provedor de credenciais mostra como criar e autenticar identidades de usuários.

O CognitoAuthentication biblioteca de extensão exemplo mostra como usar a biblioteca de CognitoAuthentication extensões para autenticar grupos de usuários do Amazon Cognito.

Tópicos

- Provedor de credenciais do Amazon Cognito
- Exemplos de bibliotecas CognitoAuthentication de extensões da Amazon

Provedor de credenciais do Amazon Cognito

> **Note**
>
> As informações neste tópico são específicas para projetos baseados no.NET Framework e na AWS SDK para .NET versão 3.3 e anteriores.

`Amazon.CognitoIdentity.CognitoAWSCredentials`, encontrado no AWSSDK. CognitoIdentity NuGetpackage, é um objeto de credenciais que usa o Amazon Cognito e AWS Security Token Service o AWS STS() para recuperar credenciais para fazer chamadas. AWS

A primeira etapa da configuração do `CognitoAWSCredentials` é criar um "grupo de identidades". Um grupo de identidades é um armazenamento de informações de identidades de usuários específico para sua conta. As informações podem ser recuperadas entre plataformas, dispositivos e sistemas operacionais de clientes, para que, se um usuário começar a usar o aplicativo em um telefone e depois mudar para um tablet, as informações mantidas do aplicativo ainda estejam disponíveis para esse usuário. É possível criar um novo banco de identidades no console do Amazon Cognito. Se você está usando o console, ele também oferece outras informações de que você precisa:

- Seu número de conta – um número de 12 dígitos, como 123456789012, que é exclusivo para sua conta.
- O ARN da função não autenticada – uma função que os usuários não autenticados assumirão. Por exemplo, essa função pode fornecer permissões somente leitura aos seus dados.
- O ARN da função autenticada – uma função que os usuários autenticados assumirão. Essa função pode fornecer permissões mais amplas aos seus dados.

Configurar o Cognito AWSCredentials

O código de exemplo a seguir mostra como configurar o `CognitoAWSCredentials`, que pode ser usado para fazer uma chamada para o Amazon S3 como usuário não autenticado. Isso permite que você faça chamadas com apenas uma quantidade mínima de dados necessária para autenticar o usuário. As permissões de usuários são controladas pela função para que seja possível configurar o acesso conforme necessário.

```
CognitoAWSCredentials credentials = new CognitoAWSCredentials(
    accountId,        // Account number
    identityPoolId,   // Identity pool ID
    unAuthRoleArn,    // Role for unauthenticated users
    null,             // Role for authenticated users, not set
    region);
using (var s3Client = new AmazonS3Client(credentials))
{
    s3Client.ListBuckets();
}
```

Use AWS como um usuário não autenticado

O exemplo de código a seguir mostra como você pode começar a usar AWS como usuário não autenticado, depois se autenticar pelo Facebook e atualizar as credenciais para usar as credenciais do Facebook. Usando essa abordagem, você pode conceder capacidades diferentes a usuários autenticados por meio da função autenticada. Por exemplo, você pode ter um aplicativo de telefone que permite que os usuários visualizem conteúdo anonimamente, mas lhes permite publicar se eles estão conectados com um ou mais dos provedores configurados.

```
CognitoAWSCredentials credentials = new CognitoAWSCredentials(
    accountId, identityPoolId,
    unAuthRoleArn,    // Role for unauthenticated users
    authRoleArn,      // Role for authenticated users
    region);
using (var s3Client = new AmazonS3Client(credentials))
{
    // Initial use will be unauthenticated
    s3Client.ListBuckets();

    // Authenticate user through Facebook
    string facebookToken = GetFacebookAuthToken();
```

```
    // Add Facebook login to credentials. This clears the current AWS credentials
    // and retrieves new AWS credentials using the authenticated role.
    credentials.AddLogin("graph.facebook.com", facebookAccessToken);

    // This call is performed with the authenticated role and credentials
    s3Client.ListBuckets();
}
```

O objeto `CognitoAWSCredentials` oferecerá ainda mais funcionalidades se você usá-lo com o `AmazonCognitoSyncClient` que faz parte do AWS SDK para .NET. Se você estiver usando `AmazonCognitoSyncClient` e `CognitoAWSCredentials`, não é necessário especificar as propriedades `IdentityPoolId` e `IdentityId` ao fazer chamadas com o `AmazonCognitoSyncClient`. Essas propriedades são preenchidas automaticamente em `CognitoAWSCredentials`. O exemplo de código a seguir ilustra isso, bem como um evento que notifica você sempre que o `IdentityId` para `CognitoAWSCredentials` é alterado. O `IdentityId` pode ser alterado em alguns casos, como na alteração de um usuário não autenticado para um autenticado.

```
CognitoAWSCredentials credentials = GetCognitoAWSCredentials();

// Log identity changes
credentials.IdentityChangedEvent += (sender, args) =>
{
    Console.WriteLine("Identity changed: [{0}] => [{1}]", args.OldIdentityId,
 args.NewIdentityId);
};

using (var syncClient = new AmazonCognitoSyncClient(credentials))
{
    var result = syncClient.ListRecords(new ListRecordsRequest
    {
        DatasetName = datasetName
        // No need to specify these properties
        //IdentityId = "...",
        //IdentityPoolId = "..."
    });
}
```

Exemplos de bibliotecas CognitoAuthentication de extensões da Amazon

> **Note**
>
> As informações neste tópico são específicas para projetos baseados no.NET Framework e na AWS SDK para .NET versão 3.3 e anteriores.

A biblioteca CognitoAuthentication de extensões, encontrada em Amazon.Extensions.CognitoAuthentication NuGet pacote, simplifica o processo de autenticação dos grupos de usuários do Amazon Cognito para desenvolvedores do.NET Core e Xamarin. A biblioteca foi criada com base na API do provedor de identidade do Amazon Cognito para criar e enviar chamadas de API de autenticação de usuários.

Usando a biblioteca CognitoAuthentication de extensões

O Amazon Cognito tem alguns valores `AuthFlow` e `ChallengeName` integrados para que um fluxo de autenticação padrão valide o nome de usuário e a senha pelo protocolo SRP (Secure Remote Password). Para obter mais informações sobre o fluxo de autenticação, consulte Amazon Cognito User Pool Authentication Flow (Fluxo de autenticação de grupos de usuários do Amazon Cognito).

Os exemplos a seguir exigem estas instruções `using`:

```
// Required for all examples
using System;
using Amazon;
using Amazon.CognitoIdentity;
using Amazon.CognitoIdentityProvider;
using Amazon.Extensions.CognitoAuthentication;
using Amazon.Runtime;
// Required for the GetS3BucketsAsync example
using Amazon.S3;
using Amazon.S3.Model;
```

Usando a autenticação básica

Crie um AmazonCognitoIdentityProviderClientusando Anonymous AWSCredentials, que não exige solicitações assinadas. Você não precisa fornecer uma região. O código subjacente chamará `FallbackRegionFactory.GetRegionEndpoint()` se uma região não for fornecida. Crie objetos

`CognitoUserPool` e `CognitoUser`. Chame o método `StartWithSrpAuthAsync` com um `InitiateSrpAuthRequest` que contenha a senha do usuário.

```
public static async void GetCredsAsync()
{
    AmazonCognitoIdentityProviderClient provider =
        new AmazonCognitoIdentityProviderClient(new
 Amazon.Runtime.AnonymousAWSCredentials());
    CognitoUserPool userPool = new CognitoUserPool("poolID", "clientID", provider);
    CognitoUser user = new CognitoUser("username", "clientID", userPool, provider);
    InitiateSrpAuthRequest authRequest = new InitiateSrpAuthRequest()
    {
        Password = "userPassword"
    };

    AuthFlowResponse authResponse = await
 user.StartWithSrpAuthAsync(authRequest).ConfigureAwait(false);
    accessToken = authResponse.AuthenticationResult.AccessToken;

}
```

Autenticar com desafios

Continuar o fluxo de autenticação com desafios, como com NewPasswordRequired e Multi-Factor Authentication (MFA), também é mais simples. Os únicos requisitos são os CognitoAuthentication objetos, a senha do usuário para o SRP e as informações necessárias para o próximo desafio, que são adquiridas após a solicitação do usuário para inseri-la. O código a seguir mostra uma forma de verificar o tipo de desafio e obter as respostas apropriadas para MFA e NewPasswordRequired desafios durante o fluxo de autenticação.

Faça uma solicitação de autenticação básica como antes, e `await` um `AuthFlowResponse`. Quando a resposta for recebida, percorra o objeto retornado `AuthenticationResult`.
Se o tipo de `ChallengeName` for `NEW_PASSWORD_REQUIRED`, chame o método `RespondToNewPasswordRequiredAsync`.

```
public static async void GetCredsChallengesAsync()
{
    AmazonCognitoIdentityProviderClient provider =
        new AmazonCognitoIdentityProviderClient(new
 Amazon.Runtime.AnonymousAWSCredentials());
    CognitoUserPool userPool = new CognitoUserPool("poolID", "clientID", provider);
    CognitoUser user = new CognitoUser("username", "clientID", userPool, provider);
```

```
    InitiateSrpAuthRequest authRequest = new InitiateSrpAuthRequest(){
        Password = "userPassword"
    };

    AuthFlowResponse authResponse = await
user.StartWithSrpAuthAsync(authRequest).ConfigureAwait(false);

    while (authResponse.AuthenticationResult == null)
    {
        if (authResponse.ChallengeName == ChallengeNameType.NEW_PASSWORD_REQUIRED)
        {
            Console.WriteLine("Enter your desired new password:");
            string newPassword = Console.ReadLine();

            authResponse = await user.RespondToNewPasswordRequiredAsync(new
RespondToNewPasswordRequiredRequest()
            {
                SessionID = authResponse.SessionID,
                NewPassword = newPassword
            });
            accessToken = authResponse.AuthenticationResult.AccessToken;
        }
        else if (authResponse.ChallengeName == ChallengeNameType.SMS_MFA)
        {
            Console.WriteLine("Enter the MFA Code sent to your device:");
            string mfaCode = Console.ReadLine();

            AuthFlowResponse mfaResponse = await user.RespondToSmsMfaAuthAsync(new
RespondToSmsMfaRequest()
            {
                SessionID = authResponse.SessionID,
                MfaCode = mfaCode

            }).ConfigureAwait(false);
            accessToken = authResponse.AuthenticationResult.AccessToken;
        }
        else
        {
            Console.WriteLine("Unrecognized authentication challenge.");
            accessToken = "";
            break;
        }
    }
```

```
    if (authResponse.AuthenticationResult != null)
    {
        Console.WriteLine("User successfully authenticated.");
    }
    else
    {
        Console.WriteLine("Error in authentication process.");
    }

}
```

Use AWS recursos após a autenticação

Depois que um usuário é autenticado usando a CognitoAuthentication biblioteca, a próxima etapa é permitir que o usuário acesse os AWS recursos apropriados. Para fazer isso, crie um banco de identidades por meio do console de identidades federadas do Amazon Cognito. Ao especificar o grupo de usuários do Amazon Cognito que você criou como provedor, usando seu poolID e clientID, você pode permitir que grupo de usuários do Amazon Cognito acesse os recursos AWS conectados à sua conta. Você também pode especificar diferentes funções para permitir que usuários autenticados e não autenticados acessem recursos diferentes. Você pode alterar essas regras no console do IAM, em que você pode adicionar ou remover permissões no campo Action (Ação) da política anexada da função. Em seguida, usando o grupo de identidades, o grupo de usuários e as informações de usuário do Amazon Cognito apropriados, você pode fazer chamadas para diferentes AWS recursos. O exemplo a seguir mostra um usuário autenticado com SRP acessando os diferentes buckets do Amazon S3 permitidos pela função do grupo de identidades associado

```
public async void GetS3BucketsAsync()
{
    var provider = new AmazonCognitoIdentityProviderClient(new
 AnonymousAWSCredentials());
    CognitoUserPool userPool = new CognitoUserPool("poolID", "clientID", provider);
    CognitoUser user = new CognitoUser("username", "clientID", userPool, provider);

    string password = "userPassword";

    AuthFlowResponse context = await user.StartWithSrpAuthAsync(new
 InitiateSrpAuthRequest()
    {
        Password = password
    }).ConfigureAwait(false);
```

```
    CognitoAWSCredentials credentials =
        user.GetCognitoAWSCredentials("identityPoolID", RegionEndpoint.<
 YourIdentityPoolRegion >);

    using (var client = new AmazonS3Client(credentials))
    {
        ListBucketsResponse response =
            await client.ListBucketsAsync(new
 ListBucketsRequest()).ConfigureAwait(false);

        foreach (S3Bucket bucket in response.Buckets)
        {
            Console.WriteLine(bucket.BucketName);
        }
    }
}
```

Opções adicionais de autenticação

Além do SRP e do MFA NewPasswordRequired, CognitoAuthentication a biblioteca de extensões oferece um fluxo de autenticação mais fácil para:

- Personalizado – iniciar com uma chamada para `StartWithCustomAuthAsync(InitiateCustomAuthRequest customRequest)`
- RefreshToken - Inicie com uma chamada para `StartWithRefreshTokenAuthAsync(InitiateRefreshTokenAuthRequest refreshTokenRequest)`
- RefreshTokenSRP - Inicie com uma chamada para `StartWithRefreshTokenAuthAsync(InitiateRefreshTokenAuthRequest refreshTokenRequest)`
- AdminNoSRP - Inicie com uma chamada para `StartWithAdminNoSrpAuthAsync(InitiateAdminNoSrpAuthRequest adminAuthRequest)`

Chame o método adequado de acordo com o fluxo que você deseja. Em seguida, continue abordando o usuário com desafios à medida que são apresentados nos objetos `AuthFlowResponse` de cada chamada de método. Além disso, chame o método de

resposta adequado, como `RespondToSmsMfaAuthAsync` para desafios de MFA e `RespondToCustomAuthAsync` para desafios personalizados.

Como usar os bancos de dados NoSQL Amazon DynamoDB

> **Note**
>
> Os modelos de programação nesses tópicos estão presentes tanto no .NET Framework quanto no .NET (Core), mas as convenções de chamada diferem, sejam síncronas ou assíncronas.

O AWS SDK para .NET suporta o Amazon DynamoDB, que é um serviço rápido de banco de dados NoSQL oferecido pela. AWS O SDK fornece três módulos de programação para comunicar-se com o DynamoDB: o modelo de baixo nível, o modelo de documento e o modelo de persistência de objetos.

As informações a seguir apresentam esses modelos e seus APIs, fornecem exemplos de como e quando usá-los e fornecem links para recursos adicionais de programação do DynamoDB no. AWS SDK para .NET

Modelo de baixo nível

O modelo de programação de baixo nível envolve chamadas diretas para o serviço do DynamoDB. Você acessa esse modelo por meio do namespace DBv2 Amazon.Dynamo.

Dos três modelos, o de baixo nível é o que exige que você escreva mais código. Por exemplo, você deve converter tipos de dados .NET para seus equivalentes no DynamoDB. Contudo, esse modelo fornece acesso à maioria dos recursos.

Os exemplos a seguir mostram como usar o modelo de baixo nível para criar ou modificar uma tabela e inserir itens em uma tabela no DynamoDB.

Criar uma tabela

No exemplo a seguir, você cria uma tabela usando o método `CreateTable` da classe `AmazonDynamoDBClient`. O método `CreateTable` usa uma instância da classe `CreateTableRequest` que contém características como nomes de atributo do item obrigatório, definição de chave primária e a capacidade de transferência. O método `CreateTable` retorna uma instância da classe `CreateTableResponse`.

```
// using Amazon.DynamoDBv2;
```

```
// using Amazon.DynamoDBv2.Model;

var client = new AmazonDynamoDBClient();

Console.WriteLine("Getting list of tables");
List<string> currentTables = client.ListTables().TableNames;
Console.WriteLine("Number of tables: " + currentTables.Count);
if (!currentTables.Contains("AnimalsInventory"))
{
    var request = new CreateTableRequest
    {
        TableName = "AnimalsInventory",
        AttributeDefinitions = new List<AttributeDefinition>
      {
        new AttributeDefinition
        {
          AttributeName = "Id",
          // "S" = string, "N" = number, and so on.
          AttributeType = "N"
        },
        new AttributeDefinition
        {
          AttributeName = "Type",
          AttributeType = "S"
        }
      },
        KeySchema = new List<KeySchemaElement>
      {
        new KeySchemaElement
        {
          AttributeName = "Id",
          // "HASH" = hash key, "RANGE" = range key.
          KeyType = "HASH"
        },
        new KeySchemaElement
        {
          AttributeName = "Type",
          KeyType = "RANGE"
        },
      },
        ProvisionedThroughput = new ProvisionedThroughput
        {
            ReadCapacityUnits = 10,
            WriteCapacityUnits = 5
```

```
        },
    };

    var response = client.CreateTable(request);

    Console.WriteLine("Table created with request ID: " +
      response.ResponseMetadata.RequestId);
}
```

Verificação de que uma tabela está pronta para ser modificada

Antes que você possa alterar ou modificar uma tabela, esta tem de estar pronta para modificação. O exemplo a seguir mostra como usar o modelo de baixo nível para verificar se uma tabela do DynamoDB está pronta. Neste exemplo, a tabela de destino a ser verificada é mencionada pelo método `DescribeTable` da classe `AmazonDynamoDBClient`. A cada cinco segundos, o código verifica o valor da propriedade `TableStatus` da tabela. Quando o status é definido como `ACTIVE`, a tabela está pronta para ser modificada.

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.Model;

var client = new AmazonDynamoDBClient();
var status = "";

do
{
  // Wait 5 seconds before checking (again).
  System.Threading.Thread.Sleep(TimeSpan.FromSeconds(5));

  try
  {
    var response = client.DescribeTable(new DescribeTableRequest
    {
      TableName = "AnimalsInventory"
    });

    Console.WriteLine("Table = {0}, Status = {1}",
      response.Table.TableName,
      response.Table.TableStatus);

    status = response.Table.TableStatus;
  }
  catch (ResourceNotFoundException)
```

```
  {
    // DescribeTable is eventually consistent. So you might
    //   get resource not found.
  }

} while (status != TableStatus.ACTIVE);
```

Introdução um item em uma tabela

No exemplo a seguir, você usa o modelo de baixo nível para inserir dois itens em uma tabela do DynamoDB. Cada item é inserido pelo método `PutItem` da classe `AmazonDynamoDBClient` usando uma instância da classe `PutItemRequest`. Cada uma das duas instâncias da classe `PutItemRequest` pega o nome da tabela em que os itens serão introduzidos, com uma série de valores de atributos de item.

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.Model;

var client = new AmazonDynamoDBClient();

var request1 = new PutItemRequest
{
  TableName = "AnimalsInventory",
  Item = new Dictionary<string, AttributeValue>
  {
    { "Id", new AttributeValue { N = "1" }},
    { "Type", new AttributeValue { S = "Dog" }},
    { "Name", new AttributeValue { S = "Fido" }}
  }
};

var request2 = new PutItemRequest
{
  TableName = "AnimalsInventory",
  Item = new Dictionary<string, AttributeValue>
  {
    { "Id", new AttributeValue { N = "2" }},
    { "Type", new AttributeValue { S = "Cat" }},
    { "Name", new AttributeValue { S = "Patches" }}
  }
};

client.PutItem(request1);
```

```
client.PutItem(request2);
```

Modelo de documento

O modelo de programação de documento oferece uma maneira mais fácil para trabalhar com dados dentro do DynamoDB. Esse modelo é destinado especificamente a acessar tabelas e itens nas tabelas. Você acessa esse modelo por meio do DBv2Amazon.Dynamo. DocumentModelnamespace.

Em comparação com o modelo de programação de baixo nível, o modelo de documento é mais fácil de codificar em relação aos dados do DynamoDB. Por exemplo, não é necessário converter tantos tipos de dados .NET em seus equivalentes no DynamoDB. Contudo, esse modelo não fornece acesso a tantos recursos como o modelo de programação de baixo nível. Por exemplo, você pode usar esse modelo para criar, recuperar, atualizar e excluir itens nas tabelas. No entanto, para criar as tabelas, é preciso usar o módulo de baixo nível. Em comparação com o modelo de persistência de objetos, este modelo exige que você grave mais código para armazenar, carregar e consultar objetos .NET.

Para obter mais informações sobre o modelo de programação de documentos do DynamoDB, consulte .NET: modelo de documento no Amazon DynamoDB Developer Guide.

As seções a seguir fornecem informações sobre como criar uma representação da tabela desejada do DynamoDB e exemplos sobre como usar o modelo de documento para inserir itens em tabelas e obter itens de tabelas.

Criar uma representação da tabela

Para realizar operações de dados usando o modelo de documento, você deve primeiro criar uma instância da classe `Table` que representa uma tabela específica. Há duas maneiras Principais de fazer isso:

LoadTable method

O primeiro mecanismo é usar um dos métodos estáticos `LoadTable` da classe `Table`, semelhante ao exemplo a seguir:

```
var client = new AmazonDynamoDBClient();
Table table = Table.LoadTable(client, "Reply");
```

> **Note**
>
> Embora esse mecanismo funcione, em determinadas condições, ele às vezes pode resultar em latência ou bloqueios adicionais devido aos comportamentos de inicialização a frio e de thread-pool. Para obter mais informações sobre esses comportamentos, consulte a postagem do blog Padrões aprimorados de inicialização do DynamoDB para o AWS SDK para .NET.

TableBuilder

Um mecanismo alternativo, a `TableBuilder`classe, foi introduzido na versão 3.7.203 do pacote .Dynamo AWSSDK. DBv2 NuGet Esse mecanismo pode abordar os comportamentos mencionados acima removendo certas chamadas de método implícitas, especificamente, o método `DescribeTable`. Esse mecanismo é usado de maneira semelhante ao seguinte exemplo:

```
var client = new AmazonDynamoDBClient();
var table = new TableBuilder(client, "Reply")
    .AddHashKey("Id", DynamoDBEntryType.String)
    .AddRangeKey("ReplyDateTime", DynamoDBEntryType.String)
    .AddGlobalSecondaryIndex("PostedBy-Message-index", "Author",
 DynamoDBEntryType.String, "Message", DynamoDBEntryType.String)
    .Build();
```

Para obter mais informações sobre esse mecanismo alternativo, consulte novamente a postagem do blog Padrões aprimorados de inicialização do DynamoDB para o AWS SDK para .NET.

Introdução de um item em uma tabela

No exemplo a seguir, um item é introduzido na tabela de respostas por maio do método `PutItemAsync` da classe `Table`. O método `PutItemAsync` leva uma instância da classe `Document`; a classe `Document` é simplesmente uma coleção de atributos inicializados.

```
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DocumentModel;

// Create a representation of the "Reply" table
//  by using one of the mechanisms described previously.

// Then, add a reply to the table.
```

```
var newReply = new Document();
newReply["Id"] = Guid.NewGuid().ToString();
newReply["ReplyDateTime"] = DateTime.UtcNow;
newReply["PostedBy"] = "Author1";
newReply["Message"] = "Thank you!";

await table.PutItemAsync(newReply);
```

Obtendo um item de uma tabela

No exemplo a seguir, uma resposta é recuperada pelo método `GetItemAsync` da classe `Table`. Para determinar a resposta a ser obtida, o `GetItemAsync` método usa a chave hash-and-range primária da resposta de destino.

```
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DocumentModel;

// Create a representation of the "Reply" table
//  by using one of the mechanisms described previously.

// Then, get a reply from the table
//  where "guid" is the hash key and "datetime" is the range key.
var reply = await table.GetItemAsync(guid, datetime);
Console.WriteLine("Id = " + reply["Id"]);
Console.WriteLine("ReplyDateTime = " + reply["ReplyDateTime"]);
Console.WriteLine("PostedBy = " + reply["PostedBy"]);
Console.WriteLine("Message = " + reply["Message"]);
```

O exemplo anterior converte implicitamente valores de tabela em strings para o método `WriteLine`. Você pode fazer conversões explícitas usando os vários métodos de “As[type]” da classe `DynamoDBEntry`. Por exemplo, você pode converter explicitamente o valor para o `Id` a partir do tipo de dado `Primitive` para um GUID por meio do método `AsGuid()`:

```
var guid = reply["Id"].AsGuid();
```

Modelo de persistência de objeto

O modelo de programação de persistência de objetos é especificamente projetado para armazenar, carregar e consultar objetos .NET no DynamoDB. Você acessa esse modelo por meio do DBv2Amazon.Dynamo. DataModelnamespace.

Dos três modelos, o modelo de persistência de objetos é o mais fácil de codificar sempre que estiver armazenando, carregando ou consultando dados do DynamoDB. Por exemplo, você trabalha diretamente com os tipos de dados do DynamoDB. Contudo, este modelo fornece acesso somente a operações que armazenam, carregam e consultam objetos .NET no DynamoDB. Por exemplo, você pode usar esse modelo para criar, recuperar, atualizar e excluir itens nas tabelas. No entanto, é preciso primeiro criar suas tabelas usando o modelo de baixo nível e, em seguida, usar esse modelo para mapear suas classes .NET para as tabelas.

Para obter mais informações sobre o modelo de programação de persistência de objetos do DynamoDB, consulte .NET: modelo de persistência de objetos no Guia do desenvolvedor do Amazon DynamoDB.

Os exemplos a seguir mostram como definir uma classe .NET que representa um item DynamoDB, usa uma instância da classe .NET para inserir um item no DynamoDB e usa uma instância de uma classe .NET para obter um item de uma tabela.

Definindo uma classe .NET que represente um item em uma tabela

No exemplo a seguir de uma definição de classe, o `DynamoDBTable` atributo especifica o nome da tabela, enquanto os `DynamoDBRangeKey` atributos `DynamoDBHashKey` e modelam a chave hash-and-range primária da tabela. O atributo `DynamoDBGlobalSecondaryIndexHashKey` é definido para que uma consulta de respostas de um autor específico possa ser construída.

```
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;

[DynamoDBTable("Reply")]
public class Reply
{
    [DynamoDBHashKey]
    public string Id { get; set; }

    [DynamoDBRangeKey(StoreAsEpoch = false)]
    public DateTime ReplyDateTime { get; set; }

    [DynamoDBGlobalSecondaryIndexHashKey("PostedBy-Message-Index",
        AttributeName ="PostedBy")]
    public string Author { get; set; }

    [DynamoDBGlobalSecondaryIndexRangeKey("PostedBy-Message-Index")]
    public string Message { get; set; }
```

```
}
```

Criando um contexto para o modelo de persistência de objetos

Para usar o modelo de programação de persistência de objetos para o DynamoDB, você deve criar um contexto que forneça uma conexão ao DynamoDB e permita acessar tabelas, e realizar diversas operações e executar consultas.

Contexto básico

O exemplo de código a seguir mostra como criar o contexto mais básico.

```
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;

var client = new AmazonDynamoDBClient();
var context = new DynamoDBContext(client);
```

Contexto com DisableFetchingTableMetadata propriedade

O exemplo a seguir mostra como você também pode definir a propriedade `DisableFetchingTableMetadata` da classe `DynamoDBContextConfig` para evitar chamadas implícitas ao método `DescribeTable`.

```
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;

var client = new AmazonDynamoDBClient();
var context = new DynamoDBContext(client, new DynamoDBContextConfig
{
    DisableFetchingTableMetadata = true
});
```

Se a propriedade `DisableFetchingTableMetadata` estiver definida como `false` (o padrão), conforme mostrado no primeiro exemplo, você poderá omitir os atributos que descrevem a estrutura de chave e índice dos itens da tabela da classe `Reply`. Em vez disso, esses atributos serão inferidos por meio de uma chamada implícita ao método `DescribeTable`. Se o `DisableFetchingTableMetadata` for definido como `true`, conforme mostrado no segundo exemplo, métodos do modelo de persistência de objetos, como o `SaveAsync` e o `QueryAsync` depende inteiramente dos atributos definidos na classe `Reply`. Nesse caso, uma chamada para o método `DescribeTable` não ocorre.

> **Note**
>
> Sob certas condições, as chamadas para o método `DescribeTable` às vezes podem levar a latência ou bloqueios adicionais devido aos comportamentos de inicialização a frio e thread-pool. Por esse motivo, às vezes é vantajoso evitar chamadas para esse método. Para obter mais informações sobre esses comportamentos, consulte a postagem do blog Padrões aprimorados de inicialização do DynamoDB para o AWS SDK para .NET.

Usando uma instância da classe .NET para inserir um item em uma tabela

Neste exemplo, um item é inserido pelo método `SaveAsync` da classe `DynamoDBContext`, que recebe uma instância inicializada da classe .NET que representa o item.

```
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;

// Create an appropriate context for the object persistence programming model,
//  examples of which have been described earlier.

// Create an object that represents the new item.
var reply = new Reply()
{
    Id = Guid.NewGuid().ToString(),
    ReplyDateTime = DateTime.UtcNow,
    Author = "Author1",
    Message = "Thank you!"
};

// Insert the item into the table.
await context.SaveAsync<Reply>(reply, new DynamoDBOperationConfig
{
    IndexName = "PostedBy-Message-index"
});
```

Usando uma instância de um objeto .NET para obter itens de uma tabela

Neste exemplo, uma consulta é criada para encontrar todos os registros de “Author1” usando o método `QueryAsync` da classe `DynamoDBContext`. Em seguida, os itens são recuperados por meio do método `GetNextSetAsync` de consulta.

```
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;

// Create an appropriate context for the object persistence programming model,
//  examples of which have been described earlier.

// Construct a query that finds all replies by a specific author.
var query = context.QueryAsync<Reply>("Author1", new DynamoDBOperationConfig
{
    IndexName = "PostedBy-Message-index"
});

// Display the result.
var set = await query.GetNextSetAsync();
foreach (var item in set)
{
    Console.WriteLine("Id = " + item.Id);
    Console.WriteLine("ReplyDateTime = " + item.ReplyDateTime);
    Console.WriteLine("PostedBy = " + item.Author);
    Console.WriteLine("Message = " + item.Message);
}
```

Informações adicionais sobre o modelo de persistência de objetos

Os exemplos e explicações mostrados acima às vezes incluem uma propriedade da classe `DynamoDBContext` chamada `DisableFetchingTableMetadata`. Essa propriedade, que foi introduzida na versão 3.7.203 do DBv2 NuGet pacote AWSSDK .Dynamo, permite que você evite certas condições que podem causar latência adicional ou bloqueios devido aos comportamentos de inicialização a frio e pool de threads. Para obter mais informações, consulte a postagem de blog Padrões aprimorados de inicialização do Dynamo DB para o AWS SDK para .NET.

A seguir, algumas informações adicionais sobre essa propriedade.

- Essa propriedade pode ser definida globalmente em seu arquivo `app.config` ou `web.config` se estiver usando o .NET Framework.
- Essa propriedade pode ser definida globalmente usando a classe `AWSConfigsDynamoDB`, como mostra o exemplo a seguir.

```
// Set the DisableFetchingTableMetadata property globally
// before constructing any context objects.
AWSConfigsDynamoDB.Context.DisableFetchingTableMetadata = true;
```

```
var client = new AmazonDynamoDBClient();
var context = new DynamoDBContext(client);
```

- Em alguns casos, você não pode adicionar atributos do DynamoDB a uma classe .NET, por exemplo, se a classe estiver definida em uma dependência. Nesses casos, ainda é possível aproveitar a propriedade `DisableFetchingTableMetadata`. Para fazer isso, use a classe `TableBuilder` além da propriedade `DisableFetchingTableMetadata`. A `TableBuilder` classe também foi introduzida na versão 3.7.203 do pacote .Dynamo AWSSDK. DBv2 NuGet

```
// Set the DisableFetchingTableMetadata property globally
// before constructing any context objects.
AWSConfigsDynamoDB.Context.DisableFetchingTableMetadata = true;

var client = new AmazonDynamoDBClient();
var context = new DynamoDBContext(client);

var table = new TableBuilder(client, "Reply")
    .AddHashKey("Id", DynamoDBEntryType.String)
    .AddRangeKey("ReplyDateTime", DynamoDBEntryType.String)
    .AddGlobalSecondaryIndex("PostedBy-Message-index", "Author",
 DynamoDBEntryType.String,
        "Message", DynamoDBEntryType.String)
    .Build();

// This registers the "Reply" table we constructed via the builder.
context.RegisterTableDefinition(table);

// Now operations like this will work,
// even if the Reply class was not annotated with this index.
var query = context.QueryAsync<Reply>("Author1", new DynamoDBOperationConfig()
{
    IndexName = "PostedBy-Message-index"
});
```

Mais informações

Usando o AWS SDK para .NET para programar o DynamoDB, informações e exemplos

- DynamoDB APIs
- Kickoff do DynamoDB Series

- DynamoDB Series – Modelo de documento
- DynamoDB Series – Esquemas de conversão
- DynamoDB Series – Modelo de persistência de objetos
- DynamoDB Series – Expressões
- Usando expressões com o Amazon DynamoDB e o AWS SDK para .NET
- Suporte para JSON no Amazon DynamoDB

Informações e exemplos do modelo de baixo nível

- Trabalhando com tabelas usando a API AWS SDK para .NET de baixo nível
- Trabalhando com itens usando a API AWS SDK para .NET de baixo nível
- Consultando tabelas usando a API de AWS SDK para .NET baixo nível
- Digitalizando tabelas usando a API AWS SDK para .NET de baixo nível
- Trabalhando com índices secundários locais usando a API AWS SDK para .NET de baixo nível
- Trabalhando com índices secundários globais usando a API AWS SDK para .NET de baixo nível

Informações e exemplos do modelo de documentos

- Tipos de dados do DynamoDB
- Dínamo DBEntry
- .NET: Modelo de documento

Informações e exemplos de modelo de persistência de objetos

- .NET: Modelo de persistência de objeto

Outras informações úteis

- Consulte Integração AWS com o.NET Aspire para acessar informações sobre o desenvolvimento com o Amazon DynamoDB local por meio do .NET Aspire.

Tópicos

- Usando expressões com o Amazon DynamoDB e o AWS SDK para .NET

- Suporte para JSON no Amazon DynamoDB

Usando expressões com o Amazon DynamoDB e o AWS SDK para .NET

> **Note**
>
> As informações neste tópico são específicas para projetos baseados no.NET Framework e na AWS SDK para .NET versão 3.3 e anteriores.

Os exemplos de código a seguir demonstram como usar o para AWS SDK para .NET programar o DynamoDB com expressões. As expressões denotam os atributos que você deseja ler de um item na tabela do DynamoDB. Você também usa expressões ao gravar um item, de forma a indicar quaisquer condições devam ser atendidas (também conhecido como atualização condicional) e como os atributos devem ser atualizados. Alguns exemplos de atualização substituem o atributo por um valor novo ou adicionam novos dados a uma lista ou um mapa. Para obter mais informações, consulte Reading and Writing Items Using Expressions (Leitura e gravação de itens usando expressões).

Tópicos

- Dados de exemplo
- Obtenha um único item usando expressões e a chave principal do item
- Obtenha vários itens usando expressões e a chave principal da tabela
- Obtenha vários itens usando expressões e outros atributos do item
- Imprimir um item
- Criar ou substituir um item usando expressões
- Atualização de um item usando expressões
- Exclusão de um item usando expressões
- Mais informações

Dados de exemplo

Os exemplos de código neste tópico dependem dos dois itens de exemplo a seguir em uma tabela do DynamoDB chamada `ProductCatalog`. Esses itens descrevem as informações sobre entradas do produto em um catálogo fictício de loja de bicicleta. Esses itens são baseados no exemplo

fornecido em Estudo de caso: um ProductCatalog item. Os descritores de tipos de dados, como `BOOL`, `L`, `M`, `N`, `NS`, `S` e `SS`, corresponde aos em JSON Data Format (Formato de dados JSON).

```
{
  "Id": {
    "N": "205"
  },
  "Title": {
    "S": "20-Bicycle 205"
  },
  "Description": {
    "S": "205 description"
  },
  "BicycleType": {
    "S": "Hybrid"
  },
  "Brand": {
    "S": "Brand-Company C"
  },
  "Price": {
    "N": "500"
  },
  "Gender": {
    "S": "B"
  },
  "Color": {
    "SS": [
      "Red",
      "Black"
    ]
  },
  "ProductCategory": {
    "S": "Bike"
  },
  "InStock": {
    "BOOL": true
  },
  "QuantityOnHand": {
    "N": "1"
  },
  "RelatedItems": {
    "NS": [
      "341",
```

```
      "472",
      "649"
    ]
  },
  "Pictures": {
    "L": [
      {
        "M": {
          "FrontView": {
            "S": "http://example/products/205_front.jpg"
          }
        }
      },
      {
        "M": {
          "RearView": {
            "S": "http://example/products/205_rear.jpg"
          }
        }
      },
      {
        "M": {
          "SideView": {
            "S": "http://example/products/205_left_side.jpg"
          }
        }
      }
    ]
  },
  "ProductReviews": {
    "M": {
      "FiveStar": {
        "SS": [
          "Excellent! Can't recommend it highly enough! Buy it!",
          "Do yourself a favor and buy this."
        ]
      },
      "OneStar": {
        "SS": [
          "Terrible product! Do not buy this."
        ]
      }
    }
  }
```

```
},
{
  "Id": {
    "N": "301"
  },
  "Title": {
    "S": "18-Bicycle 301"
  },
  "Description": {
    "S": "301 description"
  },
  "BicycleType": {
    "S": "Road"
  },
  "Brand": {
    "S": "Brand-Company C"
  },
  "Price": {
    "N": "185"
  },
  "Gender": {
    "S": "F"
  },
  "Color": {
    "SS": [
      "Blue",
      "Silver"
    ]
  },
  "ProductCategory": {
    "S": "Bike"
  },
  "InStock": {
    "BOOL": true
  },
  "QuantityOnHand": {
    "N": "3"
  },
  "RelatedItems": {
    "NS": [
      "801",
      "822",
      "979"
    ]
```

```
  },
  "Pictures": {
    "L": [
      {
        "M": {
          "FrontView": {
            "S": "http://example/products/301_front.jpg"
          }
        }
      },
      {
        "M": {
          "RearView": {
            "S": "http://example/products/301_rear.jpg"
          }
        }
      },
      {
        "M": {
          "SideView": {
            "S": "http://example/products/301_left_side.jpg"
          }
        }
      }
    ]
  },
  "ProductReviews": {
    "M": {
      "FiveStar": {
        "SS": [
          "My daughter really enjoyed this bike!"
        ]
      },
      "ThreeStar": {
        "SS": [
          "This bike was okay, but I would have preferred it in my color.",
       "Fun to ride."
        ]
      }
    }
  }
}
```

Obtenha um único item usando expressões e a chave principal do item

O exemplo a seguir apresenta o método `Amazon.DynamoDBv2.AmazonDynamoDBClient.GetItem` e um conjunto de expressões para obter e imprimir o item que tem um `Id` of `205`. Somente os atributos a seguir do item são retornados: `Id`, `Title`, `Description`, `Color`, `RelatedItems`, `Pictures` e `ProductReviews`.

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.Model;

var client = new AmazonDynamoDBClient();
var request = new GetItemRequest
{
  TableName = "ProductCatalog",
  ProjectionExpression = "Id, Title, Description, Color, #ri, Pictures, #pr",
  ExpressionAttributeNames = new Dictionary<string, string>
  {
    { "#pr", "ProductReviews" },
    { "#ri", "RelatedItems" }
  },
  Key = new Dictionary<string, AttributeValue>
  {
    { "Id", new AttributeValue { N = "205" } }
  },
};
var response = client.GetItem(request);

// PrintItem() is a custom function.
PrintItem(response.Item);
```

No exemplo anterior, a propriedade `ProjectionExpression` especifica os atributos a serem retornados. A propriedade `ExpressionAttributeNames` especifica o placeholder `#pr` para representar o atributo `ProductReviews` e o placeholder `#ri` para representar o atributo `RelatedItems`. A chamada para `PrintItem` refere-se uma função personalizada, como descrito em Imprimir um item.

Obtenha vários itens usando expressões e a chave principal da tabela

O exemplo a seguir caracteriza o método `Amazon.DynamoDBv2.AmazonDynamoDBClient.Query` e uma série de expressões para obter e imprimir o item que tenha um `Id` of `301`, mas somente se o valor de `Price` for maior que `150`. Somente os atributos a seguir do item são retornados: `Id`, `Title` e todos os atributos `ThreeStar` em `ProductReviews`.

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.Model;

var client = new AmazonDynamoDBClient();
var request = new QueryRequest
{
  TableName = "ProductCatalog",
  KeyConditions = new Dictionary<string,Condition>
  {
    { "Id", new Condition()
      {
        ComparisonOperator = ComparisonOperator.EQ,
        AttributeValueList = new List<AttributeValue>
        {
          new AttributeValue { N = "301" }
        }
      }
    }
  },
  ProjectionExpression = "Id, Title, #pr.ThreeStar",
  ExpressionAttributeNames = new Dictionary<string, string>
  {
    { "#pr", "ProductReviews" },
    { "#p", "Price" }
  },
  ExpressionAttributeValues = new Dictionary<string,AttributeValue>
  {
    { ":val", new AttributeValue { N = "150" } }
  },
  FilterExpression = "#p > :val"
};
var response = client.Query(request);

foreach (var item in response.Items)
{
  // Write out the first page of an item's attribute keys and values.
  // PrintItem() is a custom function.
  PrintItem(item);
  Console.WriteLine("=====");
}
```

No exemplo anterior, a propriedade `ProjectionExpression` especifica os atributos a serem retornados. A propriedade `ExpressionAttributeNames` especifica o espaço reservado `#pr`

para representar o atributo `ProductReviews` e o espaço reservado `#p` para representar o atributo `Price`. `#pr.ThreeStar` especifica para retornar somente o atributo `ThreeStar`. A propriedade `ExpressionAttributeValues` especifica o placeholder `:val` para representar o valor `150`. A propriedade `FilterExpression` especifica que `#p` (`Price`) deve ser maior que `:val` (`150`). A chamada para `PrintItem` refere-se uma função personalizada, como descrito em Imprimir um item.

Obtenha vários itens usando expressões e outros atributos do item

O exemplo a seguir apresenta o método `Amazon.DynamoDBv2.AmazonDynamoDBClient.Scan` e um conjunto de expressões para obter e imprimir todos os itens que tenham `ProductCategory` of `Bike`. Somente os atributos a seguir do item são retornados: `Id`, `Title` e todos os atributos em `ProductReviews`.

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.Model;

var client = new AmazonDynamoDBClient();
var request = new ScanRequest
{
  TableName = "ProductCatalog",
  ProjectionExpression = "Id, Title, #pr",
  ExpressionAttributeValues = new Dictionary<string,AttributeValue>
  {
    { ":catg", new AttributeValue { S = "Bike" } }
  },
  ExpressionAttributeNames = new Dictionary<string, string>
  {
    { "#pr", "ProductReviews" },
    { "#pc", "ProductCategory" }
  },
  FilterExpression = "#pc = :catg",
};
var response = client.Scan(request);

foreach (var item in response.Items)
{
  // Write out the first page/scan of an item's attribute keys and values.
  // PrintItem() is a custom function.
  PrintItem(item);
  Console.WriteLine("=====");
}
```

No exemplo anterior, a propriedade `ProjectionExpression` especifica os atributos a serem retornados. A propriedade `ExpressionAttributeNames` especifica o placeholder `#pr` para representar o atributo `ProductReviews` e o placeholder `#pc` para representar o atributo `ProductCategory`. A propriedade `ExpressionAttributeValues` especifica o placeholder `:catg` para representar o valor `Bike`. A propriedade `FilterExpression` especifica que `#pc` (`ProductCategory`) deve ser igual a `:catg` (`Bike`). A chamada para `PrintItem` refere-se uma função personalizada, como descrito em Imprimir um item.

Imprimir um item

O exemplo a seguir mostra como imprimir os atributos e os valores de um item. Este exemplo é usado nos exemplos anteriores que mostram como Obter um único item usando expressões e a chave principal do item, Obter vários itens usando expressões e a chave principal da tabela e Obter vários itens usando expressões e outros atributos do item.

```
// using Amazon.DynamoDBv2.Model;

// Writes out an item's attribute keys and values.
public static void PrintItem(Dictionary<string, AttributeValue> attrs)
{
  foreach (KeyValuePair<string, AttributeValue> kvp in attrs)
  {
    Console.Write(kvp.Key + " = ");
    PrintValue(kvp.Value);
  }
}

// Writes out just an attribute's value.
public static void PrintValue(AttributeValue value)
{
  // Binary attribute value.
  if (value.B != null)
  {
    Console.Write("Binary data");
  }
  // Binary set attribute value.
  else if (value.BS.Count > 0)
  {
    foreach (var bValue in value.BS)
    {
      Console.Write("\n  Binary data");
    }
```

```
  }
  // List attribute value.
  else if (value.L.Count > 0)
  {
    foreach (AttributeValue attr in value.L)
    {
      PrintValue(attr);
    }
  }
  // Map attribute value.
  else if (value.M.Count > 0)
  {
    Console.Write("\n");
    PrintItem(value.M);
  }
  // Number attribute value.
  else if (value.N != null)
  {
    Console.Write(value.N);
  }
  // Number set attribute value.
  else if (value.NS.Count > 0)
  {
    Console.Write("{0}", string.Join("\n", value.NS.ToArray()));
  }
  // Null attribute value.
  else if (value.NULL)
  {
    Console.Write("Null");
  }
  // String attribute value.
  else if (value.S != null)
  {
    Console.Write(value.S);
  }
  // String set attribute value.
  else if (value.SS.Count > 0)
  {
    Console.Write("{0}", string.Join("\n", value.SS.ToArray()));
  }
  // Otherwise, boolean value.
  else
  {
    Console.Write(value.BOOL);
```

```
  }

  Console.Write("\n");
}
```

No exemplo anterior, cada valor de atributo tem várias data-type-specific propriedades que podem ser avaliadas para determinar o formato correto para imprimir o atributo. Essas propriedades incluem `B`, `BOOL`, `BS`, `L`, `M`, `N`, `NS`, `NULL`, `S` e `SS`, que correspondem às no formato de dados JSON. Para propriedades como `B`, `N`, `NULL` e `S`, se a propriedade correspondente não for `null`, o atributo será o tipo de dados não `null` correspondente. Para propriedades como`BS`,,`L`, `MNS`, e`SS`, se `Count` for maior que zero, o atributo será do tipo de non-zero-value dados correspondente. Se todas as data-type-specific propriedades do atributo forem iguais `null` ou `Count` iguais a zero, o atributo corresponderá ao tipo de `BOOL` dados.

Criar ou substituir um item usando expressões

O exemplo a seguir apresenta o método `Amazon.DynamoDBv2.AmazonDynamoDBClient.PutItem` e um conjunto de expressões para atualizar o item com `Title` of `18-Bicycle 301`. Se o item não existir ainda, será adicionado um novo item.

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.Model;

var client = new AmazonDynamoDBClient();
var request = new PutItemRequest
{
  TableName = "ProductCatalog",
  ExpressionAttributeNames = new Dictionary<string, string>
  {
    { "#title", "Title" }
  },
  ExpressionAttributeValues = new Dictionary<string, AttributeValue>
  {
    { ":product", new AttributeValue { S = "18-Bicycle 301" } }
  },
  ConditionExpression = "#title = :product",
  // CreateItemData() is a custom function.
  Item = CreateItemData()
};
client.PutItem(request);
```

No exemplo anterior, a propriedade `ExpressionAttributeNames` especifica o placeholder `#title` para representar o atributo `Title`. A propriedade `ExpressionAttributeValues` especifica o placeholder `:product` para representar o valor `18-Bicycle 301`. A propriedade `ConditionExpression` especifica que `#title` (`Title`) deve ser igual a `:product` (`18-Bicycle 301`). A chamada para `CreateItemData` refere-se à seguinte função personalizada:

```
// using Amazon.DynamoDBv2.Model;

// Provides a sample item that can be added to a table.
public static Dictionary<string, AttributeValue> CreateItemData()
{
  var itemData = new Dictionary<string, AttributeValue>
  {
    { "Id", new AttributeValue { N = "301" } },
    { "Title", new AttributeValue { S = "18\" Girl's Bike" } },
    { "BicycleType", new AttributeValue { S = "Road" } },
    { "Brand" , new AttributeValue { S = "Brand-Company C" } },
    { "Color", new AttributeValue { SS = new List<string>{ "Blue", "Silver" } } },
    { "Description", new AttributeValue { S = "301 description" } },
    { "Gender", new AttributeValue { S = "F" } },
    { "InStock", new AttributeValue { BOOL = true } },
    { "Pictures", new AttributeValue { L = new List<AttributeValue>{
      { new AttributeValue { M = new Dictionary<string,AttributeValue>{
        { "FrontView", new AttributeValue { S = "http://example/
products/301_front.jpg" } } } } },
      { new AttributeValue { M = new Dictionary<string,AttributeValue>{
        { "RearView", new AttributeValue {S = "http://example/
products/301_rear.jpg" } } } } },
      { new AttributeValue { M = new Dictionary<string,AttributeValue>{
        { "SideView", new AttributeValue { S = "http://example/
products/301_left_side.jpg" } } } } }
    } } },
    { "Price", new AttributeValue { N = "185" } },
    { "ProductCategory", new AttributeValue { S = "Bike" } },
    { "ProductReviews", new AttributeValue { M = new Dictionary<string,AttributeValue>{
      { "FiveStar", new AttributeValue { SS = new List<string>{
        "My daughter really enjoyed this bike!" } } },
      { "OneStar", new AttributeValue { SS = new List<string>{
        "Fun to ride.",
        "This bike was okay, but I would have preferred it in my color." } } }
    } } },
    { "QuantityOnHand", new AttributeValue { N = "3" } },
```

```
    { "RelatedItems", new AttributeValue { NS = new List<string>{ "979", "822",
 "801" } } }
  };

  return itemData;
}
```

No exemplo anterior, um item de exemplo com dados de exemplo é retornado ao chamador. Uma série de atributos e valores correspondentes é construída usando os tipos de dados como `BOOL`, `L`, `M`, `N`, `NS`, `S` e `SS`, que correspondem aos do JSON Data Format (formato de dados JSON).

Atualização de um item usando expressões

O exemplo a seguir apresenta o método `Amazon.DynamoDBv2.AmazonDynamoDBClient.UpdateItem` e um conjunto de expressões para alterar `Title` para `18" Girl's Bike` para o item com `Id` de `301`.

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.Model;

var client = new AmazonDynamoDBClient();
var request = new UpdateItemRequest
{
  TableName = "ProductCatalog",
  Key = new Dictionary<string,AttributeValue>
  {
     { "Id", new AttributeValue { N = "301" } }
  },
  ExpressionAttributeNames = new Dictionary<string, string>
  {
    { "#title", "Title" }
  },
  ExpressionAttributeValues = new Dictionary<string, AttributeValue>
  {
    { ":newproduct", new AttributeValue { S = "18\" Girl's Bike" } }
  },
  UpdateExpression = "SET #title = :newproduct"
};
client.UpdateItem(request);
```

No exemplo anterior, a propriedade `ExpressionAttributeNames` especifica o placeholder `#title` para representar o atributo `Title`. A propriedade `ExpressionAttributeValues`

especifica o placeholder `:newproduct` para representar o valor `18" Girl's Bike`. A propriedade `UpdateExpression` especifica a alteração de `#title` (`Title`) para `:newproduct` (`18" Girl's Bike`).

Exclusão de um item usando expressões

O exemplo a seguir caracteriza o método `Amazon.DynamoDBv2.AmazonDynamoDBClient.DeleteItem` e um conjunto de expressões para excluir o item com `Id` de `301`, mas somente se o `Title` do item for `18-Bicycle 301`.

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.Model;

var client = new AmazonDynamoDBClient();
var request = new DeleteItemRequest
{
  TableName = "ProductCatalog",
  Key = new Dictionary<string,AttributeValue>
  {
     { "Id", new AttributeValue { N = "301" } }
  },
  ExpressionAttributeNames = new Dictionary<string, string>
  {
    { "#title", "Title" }
  },
  ExpressionAttributeValues = new Dictionary<string, AttributeValue>
  {
    { ":product", new AttributeValue { S = "18-Bicycle 301" } }
  },
  ConditionExpression = "#title = :product"
};
client.DeleteItem(request);
```

No exemplo anterior, a propriedade `ExpressionAttributeNames` especifica o placeholder `#title` para representar o atributo `Title`. A propriedade `ExpressionAttributeValues` especifica o placeholder `:product` para representar o valor `18-Bicycle 301`. A propriedade `ConditionExpression` especifica que `#title` (`Title`) deve ser igual a `:product` (`18-Bicycle 301`).

Mais informações

Para obter mais informações e exemplos de código, consulte:

- DynamoDB Series – Expressões
- Acesso aos atributos do item com expressões de projeção
- Uso de placeholders para nomes e valores de atributo
- Especificação de condições com expressões de condição
- Modificação de itens e atributos com expressões de atualização
- Trabalhando com itens usando a API AWS SDK para .NET de baixo nível
- Consultando tabelas usando a API de AWS SDK para .NET baixo nível
- Digitalizando tabelas usando a API AWS SDK para .NET de baixo nível
- Trabalhando com índices secundários locais usando a API AWS SDK para .NET de baixo nível
- Trabalhando com índices secundários globais usando a API AWS SDK para .NET de baixo nível

Suporte para JSON no Amazon DynamoDB

> **Note**
>
> As informações neste tópico são específicas para projetos baseados no.NET Framework e na AWS SDK para .NET versão 3.3 e anteriores.

O AWS SDK para .NET suporta dados JSON ao trabalhar com o Amazon DynamoDB. Isso permite que você obtenha dados formatados para JSON com mais facilidade e insira documentos em JSON nas tabelas do DynamoDB.

Tópicos

- Obtenha dados de uma tabela do DynamoDB em formato JSON
- Inserira os dados do formato JSON em uma tabela do DynamoDB
- Conversões do tipo de dados do DynamoDB para JSON
- Mais informações

Obtenha dados de uma tabela do DynamoDB em formato JSON

O exemplo a seguir mostra como obter dados de uma tabela em formato JSON:

```
// using Amazon.DynamoDBv2;
```

```
// using Amazon.DynamoDBv2.DocumentModel;

var client = new AmazonDynamoDBClient();
var table = Table.LoadTable(client, "AnimalsInventory");
var item = table.GetItem(3, "Horse");

var jsonText = item.ToJson();
Console.Write(jsonText);

// Output:
//   {"Name":"Shadow","Type":"Horse","Id":3}

var jsonPrettyText = item.ToJsonPretty();
Console.WriteLine(jsonPrettyText);

// Output:
//   {
//     "Name" : "Shadow",
//     "Type" : "Horse",
//     "Id"   : 3
//   }
```

No exemplo anterior, o método `ToJson` da classe `Document` converte um item da tabela em uma string formatada para JSON. O item é recuperado pelo método `GetItem` da classe `Table`. Para determinar o item a ser obtido, neste exemplo, o `GetItem` método usa a chave hash-and-range primária do item de destino. Para determinar a tabela de onde se obter o item, o método `LoadTable` da classe `Table` usa uma instância da classe `AmazonDynamoDBClient` e o nome da tabela de destino no DynamoDB.

Inserira os dados do formato JSON em uma tabela do DynamoDB

O exemplo a seguir mostra como usar o formato JSON para inserir um item em uma tabela do DynamoDB:

```
// using Amazon.DynamoDBv2;
// using Amazon.DynamoDBv2.DocumentModel;

var client = new AmazonDynamoDBClient();
var table = Table.LoadTable(client, "AnimalsInventory");
var jsonText = "{\"Id\":6,\"Type\":\"Bird\",\"Name\":\"Tweety\"}";
var item = Document.FromJson(jsonText);
```

```
table.PutItem(item);
```

No exemplo anterior, o método `FromJson` da classe `Document` converte uma string formatada para JSON em um item. O item é inserido na tabela pelo método `PutItem` da classe `Table`, que usa a instância da classe `Document` que contém o item. Para determinar a tabela na qual inserir o item, o método `LoadTable` da classe `Table` é chamado, especificando uma instância da classe `AmazonDynamoDBClient` e o nome da tabela de destino no DynamoDB.

Conversões do tipo de dados do DynamoDB para JSON

Sempre que você chamar o método `ToJson` da classe `Document` e depois nos dados JSON resultantes chamar o método `FromJson` para converter dados JSON de volta a uma instância de uma classe `Document`, alguns tipos de dados do DynamoDB não serão convertidos como esperado. Especificamente:

- Os conjuntos do DynamoDB (os tipos `SS`, `NS` e `BS`) serão convertidos em matrizes JSON.
- Conjuntos e escalares binários do DynamoDB (os tipos `B` e `BS`) serão convertidos em strings JSON codificadas para base64 ou em listas de strings.

  Nesse cenário, você deverá chamar o método `DecodeBase64Attributes` da classe `Document` para substituir os dados de JSON codificados para base64 pela representação binária correta. O exemplo a seguir substitui um atributo de item escalar binário codificado para base64 em uma instância de uma classe `Document`, de nome `Picture`, pela representação binária correta. Este exemplo também faz o mesmo pelo atributo do item do conjunto binário codificado para base64 na mesma instância da classe `Document`, de nome `RelatedPictures`:

  ```
  item.DecodeBase64Attributes("Picture", "RelatedPictures");
  ```

Mais informações

Para obter mais informações e exemplos de programação de JSON com o DynamoDB com o, consulte: AWS SDK para .NET

- Suporte a JSON pelo DynamoDB
- Atualização do Amazon DynamoDB – JSON, nível gratuito expandido, escalabilidade flexível, itens maiores

Trabalhar com o Amazon EC2

O AWS SDK para .NET suporta o Amazon EC2, que é um serviço web que fornece capacidade de computação redimensionável. Você usa essa capacidade de computação para criar e hospedar seus sistemas de software.

APIs

AWS SDK para .NET Ele fornece APIs para clientes do Amazon EC2. As APIs permitem que você trabalhe com recursos do EC2, como grupos de segurança e pares de chaves. As APIs também permitem que você controle as instâncias do Amazon EC2. Esta seção contém um pequeno número de exemplos que mostram os padrões que você pode seguir ao trabalhar com eles APIs. Para ver o conjunto completo de APIs, consulte a Referência da AWS SDK para .NET API (e vá até “Amazon.ec2").

O Amazon EC2 APIs é fornecido pelo AWSSDK NuGet pacote.EC2.

Pré-requisitos

Antes de começar, verifique se você configurou seu ambiente e configurou seu projeto. Revise também as informações em Uso da SDK.

Sobre os exemplos

Os exemplos nesta seção mostram como trabalhar com clientes do Amazon EC2 e como gerenciar as instâncias do Amazon EC2.

Os tutoriais de instâncias spot do EC2 mostram como solicitar instâncias spot do Amazon EC2. As instâncias spot permitem que você acesse a capacidade do EC2 não utilizada por um valor mais baixo que o preço sob demanda.

Tópicos

- Como trabalhar com grupos de segurança no Amazon EC2
- Trabalhar com pares de chaves do Amazon EC2
- Consulte regiões e zonas de disponibilidade (AZs) no Amazon EC2
- Trabalhar com as instâncias do Amazon EC2
- Tutorial de instância spot do Amazon EC2

Como trabalhar com grupos de segurança no Amazon EC2

No Amazon EC2, um grupo de segurança atua como um firewall virtual que controla o tráfego de rede para uma ou mais instâncias do EC2. Por padrão, o EC2 associa as instâncias a um grupo de segurança que não permite tráfego de entrada. Você pode criar um security group que permita que as instâncias do EC2 aceitem um determinado tráfego. Por exemplo, se você precisar se conectar a uma instância do EC2 em Windows, deverá configurar o security group para permitir tráfego RDP.

Para ler mais sobre grupos de segurança, consulte Grupos de segurança do Amazon EC2 no Guia de usuário do Amazon EC2.

> **Warning**
>
> O EC2-Classic foi removido em 15 de agosto de 2022. É recomendável migrar do EC2-Classic para uma VPC. Para obter mais informações, consulte a postagem do blog EC2-Classic Networking is Retiring – Here's How to Prepare.

Para obter informações sobre os pré-requisitos APIs e, consulte a seção principal (). Trabalhar com o Amazon EC2

Tópicos

- Como enumerar grupos de segurança
- Como criar grupos de segurança
- Como atualizar Grupos de Segurança

Como enumerar grupos de segurança

Este exemplo mostra como usar o AWS SDK para .NET para enumerar grupos de segurança. Se você fornecer um Amazon Private Cloud ID, o aplicativo enumera os grupos de segurança dessa VPC específica. Caso contrário, o aplicativo exibirá apenas uma lista de todos os grupos de segurança disponíveis.

As seções a seguir fornecem snippets desse exemplo. O código completo do exemplo é mostrado depois e pode ser criado e executado como está.

Tópicos

- Como enumerar seus grupos de segurança

- Código completo
- Considerações adicionais

Como enumerar seus grupos de segurança

O snippet a seguir enumera seus grupos de segurança. Ele enumera todos os grupos ou os grupos de uma VPC específica, se houver um.

O exemplo no final deste tópico mostra o snippet em uso.

```
//
// Method to enumerate the security groups
private static async Task EnumerateGroups(IAmazonEC2 ec2Client, string vpcID)
{
  // A request object, in case we need it.
  var request = new DescribeSecurityGroupsRequest();

  // Put together the properties, if needed
  if(!string.IsNullOrEmpty(vpcID))
  {
    // We have a VPC ID. Find the security groups for just that VPC.
    Console.WriteLine($"\nGetting security groups for VPC {vpcID}...\n");
    request.Filters.Add(new Filter
    {
      Name = "vpc-id",
      Values = new List<string>() { vpcID }
    });
  }

  // Get the list of security groups
  DescribeSecurityGroupsResponse response =
    await ec2Client.DescribeSecurityGroupsAsync(request);

  // Display the list of security groups.
  foreach (SecurityGroup item in response.SecurityGroups)
  {
    Console.WriteLine("Security group: " + item.GroupId);
    Console.WriteLine("\tGroupId: " + item.GroupId);
    Console.WriteLine("\tGroupName: " + item.GroupName);
    Console.WriteLine("\tVpcId: " + item.VpcId);
    Console.WriteLine();
  }
```

```
    }
```

Código completo

Esta seção mostra as referências relevantes e o código completo desse exemplo.

Referências do SDK

NuGet pacotes:

- AWSSDK.EC2

Elementos de programação:

- Namespace Amazon.EC2

  Classe Amazon EC2 Client
- Namespace Amazon.EC2.model

  Classe DescribeSecurityGroupsRequest

  Classe DescribeSecurityGroupsResponse

  Classe Filter

  Classe SecurityGroup

O Código

```
using System;
using System.Threading.Tasks;
using System.Collections.Generic;
using Amazon.EC2;
using Amazon.EC2.Model;

namespace EC2EnumerateSecGroups
{
  class Program
  {
    static async Task Main(string[] args)
    {
      // Parse the command line
```

```
       string vpcID = string.Empty;
      if(args.Length == 0)
      {
        Console.WriteLine("\nEC2EnumerateSecGroups [vpc_id]");
        Console.WriteLine("  vpc_id - The ID of the VPC for which you want to see
security groups.");
        Console.WriteLine("\nSince you specified no arguments, showing all available
security groups.");
      }
      else
      {
        vpcID = args[0];
      }

      if(vpcID.StartsWith("vpc-") || string.IsNullOrEmpty(vpcID))
      {
        // Create an EC2 client object
        var ec2Client = new AmazonEC2Client();

        // Enumerate the security groups
        await EnumerateGroups(ec2Client, vpcID);
      }
      else
      {
        Console.WriteLine("Could not find a valid VPC ID in the command-line
arguments:");
        Console.WriteLine($"{args[0]}");
      }
    }


    //
    // Method to enumerate the security groups
    private static async Task EnumerateGroups(IAmazonEC2 ec2Client, string vpcID)
    {
      // A request object, in case we need it.
      var request = new DescribeSecurityGroupsRequest();

      // Put together the properties, if needed
      if(!string.IsNullOrEmpty(vpcID))
      {
        // We have a VPC ID. Find the security groups for just that VPC.
        Console.WriteLine($"\nGetting security groups for VPC {vpcID}...\n");
        request.Filters.Add(new Filter
```

```
        {
          Name = "vpc-id",
          Values = new List<string>() { vpcID }
        });
      }

      // Get the list of security groups
      DescribeSecurityGroupsResponse response =
        await ec2Client.DescribeSecurityGroupsAsync(request);

      // Display the list of security groups.
      foreach (SecurityGroup item in response.SecurityGroups)
      {
        Console.WriteLine("Security group: " + item.GroupId);
        Console.WriteLine("\tGroupId: " + item.GroupId);
        Console.WriteLine("\tGroupName: " + item.GroupName);
        Console.WriteLine("\tVpcId: " + item.VpcId);
        Console.WriteLine();
      }
    }
  }
}
```

Considerações adicionais

- Observe que, no caso da VPC, o filtro é construído com a parte `Name` do par nome-valor definida como “vpc-id”. Esse nome vem da descrição da `Filters` propriedade da DescribeSecurityGroupsRequestclasse.

- Para obter a lista completa dos seus grupos de segurança, você também pode usar DescribeSecurityGroupsAsync sem parâmetros.

- Você pode verificar os resultados consultando a lista de grupos de segurança no console do Amazon EC2.

Como criar grupos de segurança

Este exemplo mostra como usar o AWS SDK para .NET para criar um grupo de segurança. Você pode fornecer o ID de uma VPC existente para criar um grupo de segurança para o EC2 em uma

VPC. Se você não fornecer essa ID, o novo grupo de segurança será para o EC2-Classic se sua AWS conta suportar isso.

Se você não fornecer uma VPC ID e sua AWS conta não for compatível com o EC2-Classic, o novo grupo de segurança pertencerá à VPC padrão da sua conta.

> **Warning**
>
> O EC2-Classic foi removido em 15 de agosto de 2022. É recomendável migrar do EC2-Classic para uma VPC. Para acessar mais informações, consulte a postagem do blog EC2-Classic Networking is Retiring - Here's How to Prepare.

As seções a seguir fornecem snippets desse exemplo. O código completo do exemplo é mostrado depois e pode ser criado e executado como está.

Tópicos

- Como encontrar grupos de segurança existentes
- Criar um grupo de segurança
- Código completo

Como encontrar grupos de segurança existentes

O snippet a seguir pesquisa grupos de segurança existentes com o nome fornecido na VPC fornecida.

O exemplo no final deste tópico mostra o snippet em uso.

```
//
// Method to determine if a security group with the specified name
// already exists in the VPC
private static async Task<List<SecurityGroup>> FindSecurityGroups(
  IAmazonEC2 ec2Client, string groupName, string vpcID)
{
  var request = new DescribeSecurityGroupsRequest();
  request.Filters.Add(new Filter{
    Name = "group-name",
    Values = new List<string>() { groupName }
  });
```

```
      if(!string.IsNullOrEmpty(vpcID))
        request.Filters.Add(new Filter{
          Name = "vpc-id",
          Values = new List<string>() { vpcID }
        });

      var response = await ec2Client.DescribeSecurityGroupsAsync(request);
      return response.SecurityGroups;
    }
```

Criar um grupo de segurança

O snippet a seguir cria um novo grupo de segurança se um grupo com esse nome não existir na VPC fornecida. Se nenhuma VPC for fornecida e existir um ou mais grupos com esse nome, o snippet apenas retornará a lista de grupos.

O exemplo no final deste tópico mostra o snippet em uso.

```
    //
    // Method to create a new security group (either EC2-Classic or EC2-VPC)
    // If vpcID is empty, the security group will be for EC2-Classic
    private static async Task<List<SecurityGroup>> CreateSecurityGroup(
      IAmazonEC2 ec2Client, string groupName, string vpcID)
    {
      // See if one or more security groups with that name
      // already exist in the given VPC. If so, return the list of them.
      var securityGroups = await FindSecurityGroups(ec2Client, groupName, vpcID);
      if (securityGroups.Count > 0)
      {
        Console.WriteLine(
          $"\nOne or more security groups with name {groupName} already exist.\n");
        return securityGroups;
      }

      // If the security group doesn't already exists, create it.
      var createRequest = new CreateSecurityGroupRequest{
        GroupName = groupName
      };
      if(string.IsNullOrEmpty(vpcID))
      {
        createRequest.Description = "My .NET example security group for EC2-Classic";
      }
      else
```

```
    {
      createRequest.VpcId = vpcID;
      createRequest.Description = "My .NET example security group for EC2-VPC";
    }
    CreateSecurityGroupResponse createResponse =
      await ec2Client.CreateSecurityGroupAsync(createRequest);

    // Return the new security group
    DescribeSecurityGroupsResponse describeResponse =
      await ec2Client.DescribeSecurityGroupsAsync(new DescribeSecurityGroupsRequest{
        GroupIds = new List<string>() { createResponse.GroupId }
      });
    return describeResponse.SecurityGroups;
  }
```

Código completo

Esta seção mostra as referências relevantes e o código completo desse exemplo.

Referências do SDK

NuGet pacotes:

- AWSSDK.EC2

Elementos de programação:

- Namespace Amazon.EC2

  Classe Amazon EC2 Client
- Namespace Amazon.EC2.model

  Classe CreateSecurityGroupRequest

  Classe CreateSecurityGroupResponse

  Classe DescribeSecurityGroupsRequest

  Classe DescribeSecurityGroupsResponse

  Classe Filter

  Classe SecurityGroup

O código

```
using System;
using System.Threading.Tasks;
using System.Collections.Generic;
using Amazon.EC2;
using Amazon.EC2.Model;

namespace EC2CreateSecGroup
{
  // = = =
```

```
      foreach(var group in securityGroups)
      {
        Console.WriteLine($"\nGroupName: {group.GroupName}");
        Console.WriteLine($"GroupId: {group.GroupId}");
        Console.WriteLine($"Description: {group.Description}");
        Console.WriteLine($"VpcId (if any): {group.VpcId}");
      }
    }


    //
    // Method to create a new security group (either EC2-Classic or EC2-VPC)
    // If vpcID is empty, the security group will be for EC2-Classic
    private static async Task<List<SecurityGroup>> CreateSecurityGroup(
      IAmazonEC2 ec2Client, string groupName, string vpcID)
    {
      // See if one or more security groups with that name
      // already exist in the given VPC. If so, return the list of them.
      var securityGroups = await FindSecurityGroups(ec2Client, groupName, vpcID);
      if (securityGroups.Count > 0)
      {
        Console.WriteLine(
          $"\nOne or more security groups with name {groupName} already exist.\n");
        return securityGroups;
      }

      // If the security group doesn't already exists, create it.
      var createRequest = new CreateSecurityGroupRequest{
        GroupName = groupName
      };
      if(string.IsNullOrEmpty(vpcID))
      {
        createRequest.Description = "Security group for .NET code example (no VPC
specified)";
      }
      else
      {
        createRequest.VpcId = vpcID;
        createRequest.Description = "Security group for .NET code example (VPC: " +
vpcID + ")";
      }
      CreateSecurityGroupResponse createResponse =
        await ec2Client.CreateSecurityGroupAsync(createRequest);
```

```
      // Return the new security group
      DescribeSecurityGroupsResponse describeResponse =
        await ec2Client.DescribeSecurityGroupsAsync(new DescribeSecurityGroupsRequest{
          GroupIds = new List<string>() { createResponse.GroupId }
        });
      return describeResponse.SecurityGroups;
    }



    //
    // Method to determine if a security group with the specified name
    // already exists in the VPC
    private static async Task<List<SecurityGroup>> FindSecurityGroups(
      IAmazonEC2 ec2Client, string groupName, string vpcID)
    {
      var request = new DescribeSecurityGroupsRequest();
      request.Filters.Add(new Filter{
        Name = "group-name",
        Values = new List<string>() { groupName }
      });
      if(!string.IsNullOrEmpty(vpcID))
        request.Filters.Add(new Filter{
          Name = "vpc-id",
          Values = new List<string>() { vpcID }
        });

      var response = await ec2Client.DescribeSecurityGroupsAsync(request);
      return response.SecurityGroups;
    }



    //
    // Command-line help
    private static void PrintHelp()
    {
      Console.WriteLine(
        "\nUsage: EC2CreateSecGroup -g <group-name> [-v <vpc-id>]" +
        "\n  -g, --group-name: The name you would like the new security group to have."
+
        "\n  -v, --vpc-id: The ID of a VPC to which the new security group will
belong." +
        "\n     If vpc-id isn't present, the security group will be" +
        "\n     for EC2-Classic (if your AWS account supports this)" +
        "\n     or will use the default VCP for EC2-VPC.");
```

```
    }
  }


  // = = =
```

```
        {
          parsedArgs.Add("--NoKey" + n.ToString(), args[i++]);
          n++;
        }
      }

      return parsedArgs;
    }

    //
    // Method to get an argument from the parsed command-line arguments
    //
    // Parameters:
    // - parsedArgs: The Dictionary object returned from the Parse() method (shown
 above).
    // - defaultValue: The default string to return if the specified key isn't in
 parsedArgs.
    // - keys: An array of keys to look for in parsedArgs.
    public static string GetArgument(
      Dictionary<string,string> parsedArgs, string defaultReturn, params string[] keys)
    {
      string retval = null;
      foreach(var key in keys)
        if(parsedArgs.TryGetValue(key, out retval)) break;
      return retval ?? defaultReturn;
    }

    //
    // Method to exit the application with an error.
    public static void ErrorExit(string msg, int code=1)
    {
      Console.WriteLine("\nError");
      Console.WriteLine(msg);
      Environment.Exit(code);
    }
  }

}
```

Como atualizar Grupos de Segurança

Este exemplo mostra como usar o AWS SDK para .NET para adicionar uma regra a um grupo de segurança. Em particular, o exemplo adiciona uma regra para permitir tráfego de entrada em

uma determinada porta TCP, que pode ser usada, por exemplo, para conexões remotas com uma instância do EC2. O aplicativo usa a ID de um grupo de segurança existente, um endereço IP (ou intervalo de endereços) no formato CIDR e, opcionalmente, um número de porta TCP. Em seguida, ele adiciona uma regra de entrada ao grupo de segurança fornecido.

> **Note**
>
> Para usar esse exemplo, você precisa de um endereço IP (ou intervalo de endereços) no formato CIDR. Consulte Considerações adicionais no final deste tópico para obter o endereço IP do seu computador local.

As seções a seguir fornecem snippets desse exemplo. O código completo do exemplo é mostrado depois e pode ser criado e executado como está.

Tópicos

- Como adicionar uma regra de entrada
- Código completo
- Considerações adicionais

Como adicionar uma regra de entrada

O trecho a seguir adiciona uma regra de entrada a um grupo de segurança para um determinado endereço IP (ou intervalo) e porta TCP.

O exemplo no final deste tópico mostra o snippet em uso.

```
//
// Method that adds a TCP ingress rule to a security group
private static async Task AddIngressRule(
  IAmazonEC2 eC2Client, string groupID, string ipAddress, int port)
{
  // Create an object to hold the request information for the rule.
  // It uses an IpPermission object to hold the IP information for the rule.
  var ingressRequest = new AuthorizeSecurityGroupIngressRequest{
    GroupId = groupID};
  ingressRequest.IpPermissions.Add(new IpPermission{
    IpProtocol = "tcp",
    FromPort = port,
    ToPort = port,
```

```
        Ipv4Ranges = new List<IpRange>() { new IpRange { CidrIp = ipAddress } }
      });

      // Create the inbound rule for the security group
      AuthorizeSecurityGroupIngressResponse responseIngress =
        await eC2Client.AuthorizeSecurityGroupIngressAsync(ingressRequest);
      Console.WriteLine($"\nNew RDP rule was written in {groupID} for {ipAddress}.");
      Console.WriteLine($"Result: {responseIngress.HttpStatusCode}");
    }
```

Código completo

Esta seção mostra as referências relevantes e o código completo desse exemplo.

Referências do SDK

NuGet pacotes:

- AWSSDK.EC2

Elementos de programação:

- Namespace Amazon.EC2

  Classe Amazon EC2 Client
- Namespace Amazon.EC2.model

  Classe AuthorizeSecurityGroupIngressRequest

  Classe AuthorizeSecurityGroupIngressResponse

  Classe IpPermission

  Classe IpRange

O código

```
using System;
using System.Threading.Tasks;
using System.Collections.Generic;
using Amazon.EC2;
using Amazon.EC2.Model;
```

```
namespace EC2AddRuleForRDP
{
  // = = =
```

```
      IAmazonEC2 eC2Client, string groupID, string ipAddress, int port)
    {
      // Create an object to hold the request information for the rule.
      // It uses an IpPermission object to hold the IP information for the rule.
      var ingressRequest = new AuthorizeSecurityGroupIngressRequest{
        GroupId = groupID};
      ingressRequest.IpPermissions.Add(new IpPermission{
        IpProtocol = "tcp",
        FromPort = port,
        ToPort = port,
        Ipv4Ranges = new List<IpRange>() { new IpRange { CidrIp = ipAddress } }
      });

      // Create the inbound rule for the security group
      AuthorizeSecurityGroupIngressResponse responseIngress =
        await eC2Client.AuthorizeSecurityGroupIngressAsync(ingressRequest);
      Console.WriteLine($"\nNew RDP rule was written in {groupID} for {ipAddress}.");
      Console.WriteLine($"Result: {responseIngress.HttpStatusCode}");
    }


    //
    // Command-line help
    private static void PrintHelp()
    {
      Console.WriteLine(
        "\nUsage: EC2AddRuleForRDP -g <group-id> -i <ip-address> [-p <port>]" +
        "\n  -g, --group-id: The ID of the security group to which you want to add the
inbound rule." +
        "\n  -i, --ip-address: An IP address or address range in CIDR format." +
        "\n  -p, --port: The TCP port number. Defaults to 3389.");
    }
  }


  // = = =
```

```
    //
    // Parameters:
    // - args: The command-line arguments passed into the application by the system.
    //
    // Returns:
    // A Dictionary with string Keys and Values.
    //
    // If a key is found without a matching value, Dictionary.Value is set to the key
    //  (including the dashes).
    // If a value is found without a matching key, Dictionary.Key is set to "--NoKeyN",
    //  where "N" represents sequential numbers.
    public static Dictionary<string,string> Parse(string[] args)
    {
      var parsedArgs = new Dictionary<string,string>();
      int i = 0, n = 0;
      while(i < args.Length)
      {
        // If the first argument in this iteration starts with a dash it's an option.
        if(args[i].StartsWith("-"))
        {
          var key = args[i++];
          var value = key;

          // Check to see if there's a value that goes with this option?
          if((i < args.Length) && (!args[i].StartsWith("-"))) value = args[i++];
          parsedArgs.Add(key, value);
        }

        // If the first argument in this iteration doesn't start with a dash, it's a
value
        else
        {
          parsedArgs.Add("--NoKey" + n.ToString(), args[i++]);
          n++;
        }
      }

      return parsedArgs;
    }

    //
    // Method to get an argument from the parsed command-line arguments
    //
    // Parameters:
```

```
    // - parsedArgs: The Dictionary object returned from the Parse() method (shown
 above).
    // - defaultValue: The default string to return if the specified key isn't in
 parsedArgs.
    // - keys: An array of keys to look for in parsedArgs.
    public static string GetArgument(
      Dictionary<string,string> parsedArgs, string defaultReturn, params string[] keys)
    {
      string retval = null;
      foreach(var key in keys)
        if(parsedArgs.TryGetValue(key, out retval)) break;
      return retval ?? defaultReturn;
    }

    //
    // Method to exit the application with an error.
    public static void ErrorExit(string msg, int code=1)
    {
      Console.WriteLine("\nError");
      Console.WriteLine(msg);
      Environment.Exit(code);
    }
  }

}
```

Considerações adicionais

- Se você não fornecer um número de porta, o aplicativo usará como padrão a porta 3389. Essa é a porta do Windows RDP, que permite que você se conecte a uma instância do EC2 executando o Windows. Se você iniciar uma instância EC2 do Linux, use a porta TCP 22 (SSH).

- Observe que o exemplo é definido como `IpProtocol` “tcp”. Os valores de `IpProtocol` podem ser encontrados na descrição da `IpProtocol` propriedade da IpPermissionclasse.

- Talvez você queira o endereço IP do computador local ao usar este exemplo. A seguir estão algumas das maneiras pelas quais você pode obter o endereço.
  - Se seu computador local (do qual você se conectará à instância do EC2) tiver um endereço IP público estático, você poderá usar um serviço para obter esse endereço. Um desses serviços é o http://checkip.amazonaws.com/. Para ler mais sobre a autorização de tráfego de entrada,

consulte Regras de grupos de segurança e Regras de grupo de segurança para diferentes casos de uso no Guia do usuário do Amazon EC2.

- Outra maneira de obter o endereço IP do seu computador local é usar o console do Amazon EC2.

  Selecione um dos seus grupos de segurança, selecione a guia Regras de entrada e escolha Editar regras de entrada. Em uma regra de entrada, abra o menu suspenso na coluna Fonte e escolha Meu IP para ver o endereço IP do seu computador local no formato CIDR. Certifique-se de cancelar a operação.

- Você pode verificar os resultados desse exemplo examinando a lista de grupos de segurança no console do Amazon EC2.

Trabalhar com pares de chaves do Amazon EC2

O Amazon EC2 utiliza criptografia de chave pública para criptografar e descriptografar as informações de login. A criptografia de chave pública usa uma chave pública para criptografar dados e, em seguida, o destinatário usa a chave privada para descriptografar os dados. As chaves pública e privada são conhecidas como par de chaves. Se quiser fazer login em uma instância do EC2, você deve especificar um par de chaves ao iniciá-la e, em seguida, fornecer a chave privada do par ao se conectar a ela.

Ao iniciar uma instância do EC2, é possível criar um par de chave para ela ou usar um que já foi usado ao iniciar outras instâncias. Para ler mais sobre pares de chaves do Amazon EC2, consulte Working with Amazon EC2 key pairs no Guia do usuário do Amazon EC2.

Para obter informações sobre os pré-requisitos APIs e, consulte a seção principal (). Trabalhar com o Amazon EC2

Tópicos

- Criar e exibir pares de chaves
- Excluir pares de chaves

Criar e exibir pares de chaves

Este exemplo mostra como usar o AWS SDK para .NET para criar um par de chaves. O aplicativo usa o nome do novo par de chaves e o nome de um arquivo PEM (com a extensão “.pem”). Ele cria o

par de chaves, grava a chave privada no arquivo PEM e exibe todos os pares de chaves disponíveis. Se você não fornecer argumentos de linha de comando, o aplicativo apenas exibirá todos os pares de chaves disponíveis.

As seções a seguir fornecem snippets desse exemplo. O código completo do exemplo é mostrado depois e pode ser criado e executado como está.

Tópicos

- Criar o par de chaves
- Exibir pares de chaves disponíveis
- Código completo
- Considerações adicionais

Criar o par de chaves

O snippet a seguir cria um par de chaves e, em seguida, armazena a chave privada no arquivo PEM fornecido.

O exemplo no final deste tópico mostra o snippet em uso.

```
//
// Method to create a key pair and save the key material in a PEM file
private static async Task CreateKeyPair(
  IAmazonEC2 ec2Client, string keyPairName, string pemFileName)
{
  // Create the key pair
  CreateKeyPairResponse response =
    await ec2Client.CreateKeyPairAsync(new CreateKeyPairRequest{
      KeyName = keyPairName
    });
  Console.WriteLine($"\nCreated new key pair: {response.KeyPair.KeyName}");

  // Save the private key in a PEM file
  using (var s = new FileStream(pemFileName, FileMode.Create))
  using (var writer = new StreamWriter(s))
  {
    writer.WriteLine(response.KeyPair.KeyMaterial);
  }
}
```

Exibir pares de chaves disponíveis

O snippet a seguir exibe uma lista dos pares de chaves disponíveis.

O exemplo no final deste tópico mostra o snippet em uso.

```
//
// Method to show the key pairs that are available
private static async Task EnumerateKeyPairs(IAmazonEC2 ec2Client)
{
  DescribeKeyPairsResponse response = await ec2Client.DescribeKeyPairsAsync();
  Console.WriteLine("Available key pairs:");
  foreach (KeyPairInfo item in response.KeyPairs)
    Console.WriteLine($"  {item.KeyName}");
}
```

Código completo

Esta seção mostra as referências relevantes e o código completo desse exemplo.

Referências do SDK

NuGet pacotes:

- AWSSDK.EC2

Elementos de programação:

- Namespace Amazon.EC2

  Classe Amazon EC2 Client
- Namespace Amazon.EC2.model

  Classe CreateKeyPairRequest

  Classe CreateKeyPairResponse

  Classe DescribeKeyPairsResponse

  Classe KeyPairInfo

O código

```
using System;
using System.Threading.Tasks;
using System.IO;
using Amazon.EC2;
using Amazon.EC2.Model;
using System.Collections.Generic;

namespace EC2CreateKeyPair
{
  // = = =
```

```
        "\nRun the command with no arguments to see help.");
    if(string.IsNullOrEmpty(pemFileName) || !pemFileName.EndsWith(".pem"))
      CommandLine.ErrorExit("\nThe PEM filename is missing or incorrect." +
        "\nRun the command with no arguments to see help.");

    // Create the key pair
    await CreateKeyPair(ec2Client, keyPairName, pemFileName);
    await EnumerateKeyPairs(ec2Client);
  }


  //
  // Method to create a key pair and save the key material in a PEM file
  private static async Task CreateKeyPair(
    IAmazonEC2 ec2Client, string keyPairName, string pemFileName)
  {
    // Create the key pair
    CreateKeyPairResponse response =
      await ec2Client.CreateKeyPairAsync(new CreateKeyPairRequest{
        KeyName = keyPairName
      });
    Console.WriteLine($"\nCreated new key pair: {response.KeyPair.KeyName}");

    // Save the private key in a PEM file
    using (var s = new FileStream(pemFileName, FileMode.Create))
    using (var writer = new StreamWriter(s))
    {
      writer.WriteLine(response.KeyPair.KeyMaterial);
    }
  }


  //
  // Method to show the key pairs that are available
  private static async Task EnumerateKeyPairs(IAmazonEC2 ec2Client)
  {
    DescribeKeyPairsResponse response = await ec2Client.DescribeKeyPairsAsync();
    Console.WriteLine("Available key pairs:");
    foreach (KeyPairInfo item in response.KeyPairs)
      Console.WriteLine($"  {item.KeyName}");
  }


  //
```

```
    // Command-line help
    private static void PrintHelp()
    {
      Console.WriteLine(
        "\nUsage: EC2CreateKeyPair -k <keypair-name> -p <pem-filename>" +
        "\n  -k, --keypair-name: The name you want to assign to the key pair." +
        "\n  -p, --pem-filename: The name of the PEM file to create, with a \".pem\"
extension.");
    }
 }


 // = = =
```

```
          // Check to see if there's a value that goes with this option?
          if((i < args.Length) && (!args[i].StartsWith("-"))) value = args[i++];
          parsedArgs.Add(key, value);
        }

        // If the first argument in this iteration doesn't start with a dash, it's a
value
        else
        {
          parsedArgs.Add("--NoKey" + n.ToString(), args[i++]);
          n++;
        }
      }

      return parsedArgs;
    }

    //
    // Method to get an argument from the parsed command-line arguments
    //
    // Parameters:
    // - parsedArgs: The Dictionary object returned from the Parse() method (shown
above).
    // - defaultValue: The default string to return if the specified key isn't in
parsedArgs.
    // - keys: An array of keys to look for in parsedArgs.
    public static string GetArgument(
      Dictionary<string,string> parsedArgs, string defaultReturn, params string[] keys)
    {
      string retval = null;
      foreach(var key in keys)
        if(parsedArgs.TryGetValue(key, out retval)) break;
      return retval ?? defaultReturn;
    }

    //
    // Method to exit the application with an error.
    public static void ErrorExit(string msg, int code=1)
    {
      Console.WriteLine("\nError");
      Console.WriteLine(msg);
      Environment.Exit(code);
    }
  }
```

```
}
```

Considerações adicionais

- Depois de executar o exemplo, você pode ver o novo par de chaves no console do Amazon EC2.

- Ao criar um par de chaves, você deve salvar a chave privada que é retornada, pois não vai poder recuperá-la depois.

Excluir pares de chaves

Este exemplo mostra como usar o AWS SDK para .NET para excluir um par de chaves. O aplicativo usa o nome de um par de chave. Ele exclui o par de chave e, em seguida, exibe todos os pares de chaves disponíveis. Se você não fornecer argumentos de linha de comando, o aplicativo apenas exibirá todos os pares de chaves disponíveis.

As seções a seguir fornecem snippets desse exemplo. O código completo do exemplo é mostrado depois e pode ser criado e executado como está.

Tópicos

- Criar pares de chaves
- Exibir pares de chaves disponíveis
- Código completo

Criar pares de chaves

O snippet a seguir exclui um par de chave.

O exemplo no final deste tópico mostra o snippet em uso.

```
    //
    // Method to delete a key pair
    private static async Task DeleteKeyPair(IAmazonEC2 ec2Client, string keyName)
    {
      await ec2Client.DeleteKeyPairAsync(new DeleteKeyPairRequest{
        KeyName = keyName});
      Console.WriteLine($"\nKey pair {keyName} has been deleted (if it existed).");
    }
```

Exibir pares de chaves disponíveis

O snippet a seguir exibe uma lista dos pares de chaves disponíveis.

O exemplo no final deste tópico mostra o snippet em uso.

```
    //
    // Method to show the key pairs that are available
    private static async Task EnumerateKeyPairs(IAmazonEC2 ec2Client)
    {
      DescribeKeyPairsResponse response = await ec2Client.DescribeKeyPairsAsync();
      Console.WriteLine("Available key pairs:");
      foreach (KeyPairInfo item in response.KeyPairs)
        Console.WriteLine($"  {item.KeyName}");
    }
```

Código completo

Esta seção mostra as referências relevantes e o código completo desse exemplo.

Referências do SDK

NuGet pacotes:

- AWSSDK.EC2

Elementos de programação:

- Namespace Amazon.EC2

  Classe Amazon EC2 Client
- Namespace Amazon.EC2.model

  Classe DeleteKeyPairRequest

  Classe DescribeKeyPairsResponse

  Classe KeyPairInfo

O código

```
using System;
```

```
using System.Threading.Tasks;
using Amazon.EC2;
using Amazon.EC2.Model;

namespace EC2DeleteKeyPair
{
  class Program
  {
    static async Task Main(string[] args)
    {
      // Create the EC2 client
      var ec2Client = new AmazonEC2Client();

      if(args.Length == 1)
      {
        // Delete a key pair (if it exists)
        await DeleteKeyPair(ec2Client, args[0]);

        // Display the key pairs that are left
        await EnumerateKeyPairs(ec2Client);
      }
      else
      {
        Console.WriteLine("\nUsage: EC2DeleteKeyPair keypair-name");
        Console.WriteLine("  keypair-name - The name of the key pair you want to
 delete.");
        Console.WriteLine("\nNo arguments specified.");
        Console.Write(
          "Do you want to see a list of the existing key pairs? ((y) or n): ");
        string response = Console.ReadLine();
        if((string.IsNullOrEmpty(response)) || (response.ToLower() == "y"))
          await EnumerateKeyPairs(ec2Client);
      }
    }


    //
    // Method to delete a key pair
    private static async Task DeleteKeyPair(IAmazonEC2 ec2Client, string keyName)
    {
      await ec2Client.DeleteKeyPairAsync(new DeleteKeyPairRequest{
        KeyName = keyName});
      Console.WriteLine($"\nKey pair {keyName} has been deleted (if it existed).");
    }
```

```
    //
    // Method to show the key pairs that are available
    private static async Task EnumerateKeyPairs(IAmazonEC2 ec2Client)
    {
      DescribeKeyPairsResponse response = await ec2Client.DescribeKeyPairsAsync();
      Console.WriteLine("Available key pairs:");
      foreach (KeyPairInfo item in response.KeyPairs)
        Console.WriteLine($"  {item.KeyName}");
    }
  }
}
```

Consulte regiões e zonas de disponibilidade (AZs) no Amazon EC2

O Amazon EC2 está hospedado em vários locais do mundo. Esses locais são compostos por regiões da e zonas de disponibilidade. Cada região é uma área geográfica separada, com vários locais isolados, conhecidos como zonas de disponibilidade.

Para ler mais sobre regiões e zonas de disponibilidade, consulte Regiões e zonas de disponibilidade no Guia do usuário do Amazon EC2.

Este exemplo mostra como usar o AWS SDK para .NET para obter detalhes sobre as regiões e zonas de disponibilidade relacionadas a um cliente EC2. O aplicativo exibe listas das regiões e zonas de disponibilidade disponíveis para um cliente EC2.

Referências do SDK

NuGet pacotes:

- AWSSDK.EC2

Elementos de programação:

- Namespace Amazon.EC2

  Classe Amazon EC2 Client
- Namespace Amazon.EC2.model

  Classe DescribeAvailabilityZonesResponse

Classe DescribeRegionsResponse

Classe AvailabilityZone

Classe Region

```
using System;
using System.Threading.Tasks;
using Amazon.EC2;
using Amazon.EC2.Model;

namespace EC2RegionsAndZones
{
  class Program
  {
    static async Task Main(string[] args)
    {
      Console.WriteLine(
        "Finding the Regions and Availability Zones available to an EC2 client...");

      // Create the EC2 client
      var ec2Client = new AmazonEC2Client();

      // Display the Regions and Availability Zones
      await DescribeRegions(ec2Client);
      await DescribeAvailabilityZones(ec2Client);
    }


    //
    // Method to display Regions
    private static async Task DescribeRegions(IAmazonEC2 ec2Client)
    {
      Console.WriteLine("\nRegions that are enabled for the EC2 client:");
      DescribeRegionsResponse response = await ec2Client.DescribeRegionsAsync();
      foreach (Region region in response.Regions)
        Console.WriteLine(region.RegionName);
    }


    //
    // Method to display Availability Zones
```

```
    private static async Task DescribeAvailabilityZones(IAmazonEC2 ec2Client)
    {
      Console.WriteLine("\nAvailability Zones for the EC2 client's region:");
      DescribeAvailabilityZonesResponse response =
        await ec2Client.DescribeAvailabilityZonesAsync();
      foreach (AvailabilityZone az in response.AvailabilityZones)
        Console.WriteLine(az.ZoneName);
    }
  }
}
```

Trabalhar com as instâncias do Amazon EC2

Você pode usar o AWS SDK para .NET para controlar instâncias do Amazon EC2 com operações como criar, iniciar e encerrar. Os tópicos nesta seção fornecem alguns exemplos de como fazer isso. Para ler mais sobre instâncias do EC2, consulte as Instâncias do Amazon EC2 no Guia do usuário do Amazon EC2.

Para obter informações sobre os pré-requisitos APIs e, consulte a seção principal (). Trabalhar com o Amazon EC2

Tópicos

- Iniciar uma instância do Amazon EC2
- Encerramento de uma instância Amazon EC2

Iniciar uma instância do Amazon EC2

Este exemplo mostra como usar o AWS SDK para .NET para iniciar uma ou mais instâncias do Amazon EC2 configuradas de forma idêntica a partir da mesma Amazon Machine Image (AMI). Ao usar várias entradas que você fornece, o aplicativo executa uma instância do EC2 e monitora a instância até que ela saia do estado “Pendente”.

Quando a instância do EC2 estiver em execução, será possível se conectar a ela remotamente, conforme descrito em (opcional) Conecte-se à instância.

> **Warning**
>
> O EC2-Classic foi removido em 15 de agosto de 2022. É recomendável migrar do EC2-Classic para uma VPC. Para acessar mais informações, consulte a postagem do blog EC2-Classic Networking is Retiring - Here's How to Prepare.

As seções a seguir fornecem snippets e outras informações para este exemplo. O código completo do exemplo é mostrado após os snippets e pode ser criado e executado como está.

Tópicos

- Colete o que você precisa
- Iniciar uma instância
- Monitorar instância
- Código completo
- Considerações adicionais
- (opcional) Conecte-se à instância
- Limpeza

Colete o que você precisa

Para iniciar uma instância do EC2, você precisará de várias coisas.

- Uma VPC onde a instância será iniciada. Se for uma instância do Windows e você estiver se conectando a ela por meio de RDP, a VPC provavelmente precisará de um gateway da Internet anexado a ela, bem como uma entrada para o gateway de Internet na tabela de rotas. Para obter mais informações, consulte Gateways da Internet no Guia do usuário da Amazon VPC.

- O ID da sub-rede na qual a instância será iniciada. Uma maneira fácil de encontrar ou criar isso é fazer login no console da Amazon VPC, mas você também pode obtê-lo programaticamente usando os métodos e. CreateSubnetAsyncDescribeSubnetsAsync

> **Note**
>
> Se você não fornecer esse parâmetro, a nova instância será executada na VPC padrão da conta.

- O ID de um grupo de segurança existente que pertence à VPC em que a instância será iniciada. Para obter mais informações, consulte Como trabalhar com grupos de segurança no Amazon EC2.

- Se você quiser se conectar à nova instância, o grupo de segurança mencionado anteriormente deve ter uma regra de entrada apropriada que permita tráfego SSH na porta 22 (instância Linux) ou tráfego RDP na porta 3389 (instância do Windows). Para obter informações sobre como fazer isso, consulte Como atualizar Grupos de Segurança, inclusive o Considerações adicionais próximo ao final desse tópico.

- A imagem de máquina da Amazon (AMI) que será usada para criar a instância. Para obter informações sobre isso AMIs, consulte Amazon Machine Images (AMIs) no Guia do usuário do Amazon EC2. Em particular, consulte Encontre uma AMI e compartilhe AMIs.

- O nome de um par de chave do EC2 existente, que é usado para se conectar à nova instância. Para obter mais informações, consulte Trabalhar com pares de chaves do Amazon EC2.

- O nome do arquivo PEM que contém a chave privada do par de chave EC2 mencionado anteriormente. O arquivo PEM é usado quando você se conecta remotamente à instância.

Iniciar uma instância

O seguinte snippet inicia uma instância do EC2.

Os exemplos no final deste tópico mostram o snippet em uso.

```
    //
    // Method to launch the instances
    // Returns a list with the launched instance IDs
    private static async Task<List<string>> LaunchInstances(
      IAmazonEC2 ec2Client, RunInstancesRequest requestLaunch)
```

```
    {
      var instanceIds = new List<string>();
      RunInstancesResponse responseLaunch =
        await ec2Client.RunInstancesAsync(requestLaunch);

      Console.WriteLine("\nNew instances have been created.");
      foreach (Instance item in responseLaunch.Reservation.Instances)
      {
        instanceIds.Add(item.InstanceId);
        Console.WriteLine($"  New instance: {item.InstanceId}");
      }

      return instanceIds;
    }
```

Monitorar instância

O snippet a seguir monitora a instância até que ela saia do estado “Pendente”.

Os exemplos no final deste tópico mostram o snippet em uso.

Consulte a InstanceStateclasse para ver os valores válidos da `Instance.State.Code` propriedade.

```
    //
    // Method to wait until the instances are running (or at least not pending)
    private static async Task CheckState(IAmazonEC2 ec2Client, List<string>
instanceIds)
    {
      Console.WriteLine(
        "\nWaiting for the instances to start." +
        "\nPress any key to stop waiting. (Response might be slightly delayed.)");

      int numberRunning;
      DescribeInstancesResponse responseDescribe;
      var requestDescribe = new DescribeInstancesRequest{
        InstanceIds = instanceIds};

      // Check every couple of seconds
      int wait = 2000;
      while(true)
      {
        // Get and check the status for each of the instances to see if it's past
pending.
```

```
        // Once all instances are past pending, break out.
        // (For this example, we are assuming that there is only one reservation.)
        Console.Write(".");
        numberRunning = 0;
        responseDescribe = await ec2Client.DescribeInstancesAsync(requestDescribe);
        foreach(Instance i in responseDescribe.Reservations[0].Instances)
        {
          // Check the lower byte of State.Code property
          // Code == 0 is the pending state
          if((i.State.Code & 255) > 0) numberRunning++;
        }
        if(numberRunning == responseDescribe.Reservations[0].Instances.Count)
          break;

        // Wait a bit and try again (unless the user wants to stop waiting)
        Thread.Sleep(wait);
        if(Console.KeyAvailable)
          break;
      }

      Console.WriteLine("\nNo more instances are pending.");
      foreach(Instance i in responseDescribe.Reservations[0].Instances)
      {
        Console.WriteLine($"For {i.InstanceId}:");
        Console.WriteLine($"  VPC ID: {i.VpcId}");
        Console.WriteLine($"  Instance state: {i.State.Name}");
        Console.WriteLine($"  Public IP address: {i.PublicIpAddress}");
        Console.WriteLine($"  Public DNS name: {i.PublicDnsName}");
        Console.WriteLine($"  Key pair name: {i.KeyName}");
      }
    }
```

Código completo

Esta seção mostra as referências relevantes e o código completo desse exemplo.

Referências do SDK

NuGet pacotes:

- AWSSDK.EC2

Elementos de programação:

- Namespace Amazon.EC2

  Classe Amazon EC2 Client

  Classe InstanceType
- Namespace Amazon.EC2.model

  Classe DescribeInstancesRequest

  Classe DescribeInstancesResponse

  Classe Instância

  Classe InstanceNetworkInterfaceSpecification

  Classe RunInstancesRequest

  Classe RunInstancesResponse

O código

```
using System;
using System.Threading;
using System.Threading.Tasks;
using System.Collections.Generic;
using Amazon.EC2;
using Amazon.EC2.Model;

namespace EC2LaunchInstance
{
  // = = =
```

```
        return;
      }

      // Get the application arguments from the parsed list
      string groupID =
        CommandLine.GetArgument(parsedArgs, null, "-g", "--group-id");
      string ami =
        CommandLine.GetArgument(parsedArgs, null, "-a", "--ami-id");
      string keyPairName =
        CommandLine.GetArgument(parsedArgs, null, "-k", "--keypair-name");
      string subnetID =
        CommandLine.GetArgument(parsedArgs, null, "-s", "--subnet-id");
      if(   (string.IsNullOrEmpty(groupID) || !groupID.StartsWith("sg-"))
         || (string.IsNullOrEmpty(ami) || !ami.StartsWith("ami-"))
         || (string.IsNullOrEmpty(keyPairName))
         || (!string.IsNullOrEmpty(subnetID) && !subnetID.StartsWith("subnet-")))
        CommandLine.ErrorExit(
          "\nOne or more of the required arguments is missing or incorrect." +
          "\nRun the command with no arguments to see help.");

      // Create an EC2 client
      var ec2Client = new AmazonEC2Client();

      // Create an object with the necessary properties
      RunInstancesRequest request = GetRequestData(groupID, ami, keyPairName,
subnetID);

      // Launch the instances and wait for them to start running
      var instanceIds = await LaunchInstances(ec2Client, request);
      await CheckState(ec2Client, instanceIds);
    }


    //
    // Method to put together the properties needed to launch the instance.
    private static RunInstancesRequest GetRequestData(
      string groupID, string ami, string keyPairName, string subnetID)
    {
      // Common properties
      var groupIDs = new List<string>() { groupID };
      var request = new RunInstancesRequest()
      {
        // The first three of these would be additional command-line arguments or
similar.
```

```
        InstanceType = InstanceType.T1Micro,
        MinCount = 1,
        MaxCount = 1,
        ImageId = ami,
        KeyName = keyPairName
      };

      // Properties specifically for EC2 in a VPC.
      if(!string.IsNullOrEmpty(subnetID))
      {
        request.NetworkInterfaces =
          new List<InstanceNetworkInterfaceSpecification>() {
            new InstanceNetworkInterfaceSpecification() {
              DeviceIndex = 0,
              SubnetId = subnetID,
              Groups = groupIDs,
              AssociatePublicIpAddress = true
            }
          };
      }

      // Properties specifically for EC2-Classic
      else
      {
        request.SecurityGroupIds = groupIDs;
      }
      return request;
    }


    //
    // Method to launch the instances
    // Returns a list with the launched instance IDs
    private static async Task<List<string>> LaunchInstances(
      IAmazonEC2 ec2Client, RunInstancesRequest requestLaunch)
    {
      var instanceIds = new List<string>();
      RunInstancesResponse responseLaunch =
        await ec2Client.RunInstancesAsync(requestLaunch);

      Console.WriteLine("\nNew instances have been created.");
      foreach (Instance item in responseLaunch.Reservation.Instances)
      {
        instanceIds.Add(item.InstanceId);
```

```
        Console.WriteLine($"  New instance: {item.InstanceId}");
      }

      return instanceIds;
    }


    //
    // Method to wait until the instances are running (or at least not pending)
    private static async Task CheckState(IAmazonEC2 ec2Client, List<string>
instanceIds)
    {
      Console.WriteLine(
        "\nWaiting for the instances to start." +
        "\nPress any key to stop waiting. (Response might be slightly delayed.)");

      int numberRunning;
      DescribeInstancesResponse responseDescribe;
      var requestDescribe = new DescribeInstancesRequest{
        InstanceIds = instanceIds};

      // Check every couple of seconds
      int wait = 2000;
      while(true)
      {
        // Get and check the status for each of the instances to see if it's past
pending.
        // Once all instances are past pending, break out.
        // (For this example, we are assuming that there is only one reservation.)
        Console.Write(".");
        numberRunning = 0;
        responseDescribe = await ec2Client.DescribeInstancesAsync(requestDescribe);
        foreach(Instance i in responseDescribe.Reservations[0].Instances)
        {
          // Check the lower byte of State.Code property
          // Code == 0 is the pending state
          if((i.State.Code & 255) > 0) numberRunning++;
        }
        if(numberRunning == responseDescribe.Reservations[0].Instances.Count)
          break;

        // Wait a bit and try again (unless the user wants to stop waiting)
        Thread.Sleep(wait);
        if(Console.KeyAvailable)
```

```
          break;
      }

      Console.WriteLine("\nNo more instances are pending.");
      foreach(Instance i in responseDescribe.Reservations[0].Instances)
      {
        Console.WriteLine($"For {i.InstanceId}:");
        Console.WriteLine($"  VPC ID: {i.VpcId}");
        Console.WriteLine($"  Instance state: {i.State.Name}");
        Console.WriteLine($"  Public IP address: {i.PublicIpAddress}");
        Console.WriteLine($"  Public DNS name: {i.PublicDnsName}");
        Console.WriteLine($"  Key pair name: {i.KeyName}");
      }
    }


    //
    // Command-line help
    private static void PrintHelp()
    {
      Console.WriteLine(
        "\nUsage: EC2LaunchInstance -g <group-id> -a <ami-id> -k <keypair-name> [-s
<subnet-id>]" +
        "\n  -g, --group-id: The ID of the security group." +
        "\n  -a, --ami-id: The ID of an Amazon Machine Image." +
        "\n  -k, --keypair-name - The name of a key pair." +
        "\n  -s, --subnet-id: The ID of a subnet. Required only for EC2 in a VPC.");
    }
  }


  // = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = =
  // Class that represents a command line on the console or terminal.
  // (This is the same for all examples. When you have seen it once, you can ignore
it.)
  static class CommandLine
  {
    //
    // Method to parse a command line of the form: "--key value" or "-k value".
    //
    // Parameters:
    // - args: The command-line arguments passed into the application by the system.
    //
```

```
    // Returns:
    // A Dictionary with string Keys and Values.
    //
    // If a key is found without a matching value, Dictionary.Value is set to the key
    //  (including the dashes).
    // If a value is found without a matching key, Dictionary.Key is set to "--NoKeyN",
    //  where "N" represents sequential numbers.
    public static Dictionary<string,string> Parse(string[] args)
    {
      var parsedArgs = new Dictionary<string,string>();
      int i = 0, n = 0;
      while(i < args.Length)
      {
        // If the first argument in this iteration starts with a dash it's an option.
        if(args[i].StartsWith("-"))
        {
          var key = args[i++];
          var value = key;

          // Check to see if there's a value that goes with this option?
          if((i < args.Length) && (!args[i].StartsWith("-"))) value = args[i++];
          parsedArgs.Add(key, value);
        }

        // If the first argument in this iteration doesn't start with a dash, it's a
value
        else
        {
          parsedArgs.Add("--NoKey" + n.ToString(), args[i++]);
          n++;
        }
      }

      return parsedArgs;
    }

    //
    // Method to get an argument from the parsed command-line arguments
    //
    // Parameters:
    // - parsedArgs: The Dictionary object returned from the Parse() method (shown
above).
    // - defaultValue: The default string to return if the specified key isn't in
parsedArgs.
```

```
    // - keys: An array of keys to look for in parsedArgs.
    public static string GetArgument(
      Dictionary<string,string> parsedArgs, string defaultReturn, params string[] keys)
    {
      string retval = null;
      foreach(var key in keys)
        if(parsedArgs.TryGetValue(key, out retval)) break;
      return retval ?? defaultReturn;
    }

    //
    // Method to exit the application with an error.
    public static void ErrorExit(string msg, int code=1)
    {
      Console.WriteLine("\nError");
      Console.WriteLine(msg);
      Environment.Exit(code);
    }
  }

}
```

Considerações adicionais

- Ao verificar o estado de uma instância do EC2, você pode adicionar um filtro à `Filter` propriedade do DescribeInstancesRequestobjeto. Usando essa técnica, você pode limitar a solicitação a determinadas instâncias; por exemplo, instâncias com uma tag específica determinada pelo usuário.

- Para resumir, algumas propriedades receberam valores típicos. Em vez disso, qualquer uma, ou todas essas propriedades, podem ser determinadas programaticamente ou por meio da entrada do usuário.

- Os valores que você pode usar para as `MaxCount` propriedades `MinCount` e do RunInstancesRequestobjeto são determinados pela zona de disponibilidade de destino e pelo número máximo de instâncias permitidas para o tipo de instância. Para obter mais informações, consulte Quantas instâncias posso executar no Amazon EC2? nas perguntas frequentes (FAQs) do Amazon EC2.

- Se você quiser usar um tipo de instância diferente deste exemplo, há alguns tipos para escolher. Para acessar mais informações, consulte Tipos de instâncias do Amazon EC2 no Guia do usuário do Amazon EC2. Consulte também Detalhes do tipo de instância e Explorador de tipos de instância.

- Você também pode anexar um perfil do IAM a uma instância ao iniciá-la. Para fazer isso, crie um IamInstanceProfileSpecificationobjeto cuja `Name` propriedade esteja definida como o nome de uma função do IAM. Em seguida, adicione esse objeto à `IamInstanceProfile` propriedade do RunInstancesRequestobjeto.

  > **Note**
  >
  > Para iniciar uma instância do EC2 com um perfil do IAM anexado, uma configuração de um usuário do IAM deve incluir determinadas permissões. Para acessar mais informações sobre as permissões necessárias, consulte Conceder permissão a um usuário para passar um perfil do IAM para uma instância no Guia do usuário do Amazon EC2.

(opcional) Conecte-se à instância

Depois que uma instância estiver sendo executada, você pode conectar-se a ela remotamente usando o cliente remoto apropriado. Para instâncias Linux e Windows, você precisa do endereço IP público ou do nome DNS público da instância. Você precisará também dos seguintes itens:

Para instâncias do Linux

Também é possível usar um cliente SSH para conectar-se à sua instância do Linux. Certifique-se de que o grupo de segurança que você usou ao iniciar a instância permita tráfego SSH na porta 22, conforme descrito em Como atualizar Grupos de Segurança.

Você também precisa da parte privada do par de chaves que você usou para iniciar a instância, ou seja, o arquivo PEM.

Para acessar mais informações, consulte Conectar-se à instância do Linux no Guia do usuário do Amazon EC2.

Para instâncias do Windows

Também é possível usar um cliente RDP para conectar-se à sua instância. Certifique-se de que o grupo de segurança que você usou ao iniciar a instância permita tráfego RDP na porta 3389, conforme descrito em Como atualizar Grupos de Segurança.

Você precisará também da senha do administrador. Você pode obter isso usando o código de exemplo a seguir, que exige o ID da instância e a parte privada do par de chave usado para iniciar a instância, ou seja, o arquivo PEM.

Para acessar mais informações, consulte Conectar-se à sua instância do Windows no Guia do usuário do Amazon EC2.

> **Warning**
>
> Esse código de exemplo retorna a senha de administrador em texto simples para sua instância.

Referências do SDK

NuGet pacotes:

- AWSSDK.EC2

Elementos de programação:

- Namespace Amazon.EC2

  Classe Amazon EC2 Client
- Namespace Amazon.EC2.model

  Classe GetPasswordDataRequest

  Classe GetPasswordDataResponse

O código

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Threading.Tasks;
```

```
using Amazon.EC2;
using Amazon.EC2.Model;

namespace EC2GetWindowsPassword
{
  // = = =
```

```
    {
      string password = string.Empty;
      GetPasswordDataResponse response =
        await ec2Client.GetPasswordDataAsync(new GetPasswordDataRequest{
          InstanceId = instanceID});
      if(response.PasswordData != null)
      {
        password = response.GetDecryptedPassword(File.ReadAllText(pemFilename));
      }
      else
      {
        Console.WriteLine($"\nThe password is not available for instance
{instanceID}.");
        Console.WriteLine($"If this is a Windows instance, the password might not be
ready.");
      }
      return password;
    }


    //
    // Command-line help
    private static void PrintHelp()
    {
      Console.WriteLine(
        "\nUsage: EC2GetWindowsPassword -i <instance-id> -p pem-filename" +
        "\n  -i, --instance-id: The name of the EC2 instance." +
        "\n  -p, --pem-filename: The name of the PEM file with the private key.");
    }
  }

  // = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = =
  // Class that represents a command line on the console or terminal.
  // (This is the same for all examples. When you have seen it once, you can ignore
it.)
  static class CommandLine
  {
    //
    // Method to parse a command line of the form: "--key value" or "-k value".
    //
    // Parameters:
    // - args: The command-line arguments passed into the application by the system.
    //
```

```
    // Returns:
    // A Dictionary with string Keys and Values.
    //
    // If a key is found without a matching value, Dictionary.Value is set to the key
    //  (including the dashes).
    // If a value is found without a matching key, Dictionary.Key is set to "--NoKeyN",
    //  where "N" represents sequential numbers.
    public static Dictionary<string,string> Parse(string[] args)
    {
      var parsedArgs = new Dictionary<string,string>();
      int i = 0, n = 0;
      while(i < args.Length)
      {
        // If the first argument in this iteration starts with a dash it's an option.
        if(args[i].StartsWith("-"))
        {
          var key = args[i++];
          var value = key;

          // Check to see if there's a value that goes with this option?
          if((i < args.Length) && (!args[i].StartsWith("-"))) value = args[i++];
          parsedArgs.Add(key, value);
        }

        // If the first argument in this iteration doesn't start with a dash, it's a
value
        else
        {
          parsedArgs.Add("--NoKey" + n.ToString(), args[i++]);
          n++;
        }
      }

      return parsedArgs;
    }

    //
    // Method to get an argument from the parsed command-line arguments
    //
    // Parameters:
    // - parsedArgs: The Dictionary object returned from the Parse() method (shown
above).
    // - defaultValue: The default string to return if the specified key isn't in
parsedArgs.
```

```
    // - keys: An array of keys to look for in parsedArgs.
    public static string GetArgument(
      Dictionary<string,string> parsedArgs, string defaultReturn, params string[] keys)
    {
      string retval = null;
      foreach(var key in keys)
        if(parsedArgs.TryGetValue(key, out retval)) break;
      return retval ?? defaultReturn;
    }

    //
    // Method to exit the application with an error.
    public static void ErrorExit(string msg, int code=1)
    {
      Console.WriteLine("\nError");
      Console.WriteLine(msg);
      Environment.Exit(code);
    }
  }

}
```

Limpeza

Quando você não precisar mais da instância do EC2, encerre-a, conforme descrito em Encerramento de uma instância Amazon EC2.

Encerramento de uma instância Amazon EC2

Quando uma ou mais instâncias do Amazon EC2 não forem mais necessárias, é possível encerrá-las.

Este exemplo mostra como usar o para AWS SDK para .NET encerrar instâncias do EC2. A entrada requer um ID de instância.

Referências do SDK

NuGet pacotes:

- AWSSDK.EC2

Elementos de programação:

- Namespace Amazon.EC2

  Classe Amazon EC2 Client

- Namespace Amazon.EC2.model

  Classe TerminateInstancesRequest

  Classe TerminateInstancesResponse

```
using System;
using System.Threading.Tasks;
using System.Collections.Generic;
using Amazon.EC2;
using Amazon.EC2.Model;

namespace EC2TerminateInstance
{
  class Program
  {
    static async Task Main(string[] args)
    {
      if((args.Length == 1) && (args[0].StartsWith("i-")))
      {
        // Terminate the instance
        var ec2Client = new AmazonEC2Client();
        await TerminateInstance(ec2Client, args[0]);
      }
      else
      {
        Console.WriteLine("\nCommand-line argument missing or incorrect.");
        Console.WriteLine("\nUsage: EC2TerminateInstance instance-ID");
        Console.WriteLine("  instance-ID - The EC2 instance you want to terminate.");
        return;
      }
    }

    //
    // Method to terminate an EC2 instance
    private static async Task TerminateInstance(IAmazonEC2 ec2Client, string
 instanceID)
    {
      var request = new TerminateInstancesRequest{
```

```
        InstanceIds = new List<string>() { instanceID }};
      TerminateInstancesResponse response =
        await ec2Client.TerminateInstancesAsync(new TerminateInstancesRequest{
          InstanceIds = new List<string>() { instanceID }
        });
      foreach (InstanceStateChange item in response.TerminatingInstances)
      {
        Console.WriteLine("Terminated instance: " + item.InstanceId);
        Console.WriteLine("Instance state: " + item.CurrentState.Name);
      }
    }
  }
}
```

Depois de executar o exemplo, é uma boa ideia fazer login no console do Amazon EC2 para verificar se a instância do EC2 foi encerrada.

Tutorial de instância spot do Amazon EC2

Este tutorial mostra como usar o AWS SDK para .NET para gerenciar instâncias spot do Amazon EC2.

Visão geral do

Instâncias spot permitem solicitar capacidade não utilizada do Amazon EC2 por um valor mais baixo que o preço sob demanda. Isso pode reduzir significativamente os custos do EC2 para aplicativos que podem ser interrompidos.

O resumo a seguir detalha como as instâncias spot são solicitadas e usadas.

1. Criação uma solicitação de instância spot, especificando o preço máximo que está disposto a pagar.
2. Quando a solicitação for atendida, execute a instância como faria com qualquer outra instância do Amazon EC2.
3. Execute a instância por quanto tempo quiser e, em seguida, encerre-a, a menos que o preço de spot mude de maneira que a instância fique encerrada para você.
4. Limpe a solicitação da Instância Spot quando não for mais necessária, para que as Instâncias Spot não sejam mais criadas.

Essa tem sido uma visão geral de alto nível das Instâncias Spot. Para ter uma melhor compreensão das instâncias spot, consulte Instâncias spot no Guia do usuário do Amazon EC2.

Sobre este tutorial

Ao seguir este tutorial, você usa o AWS SDK para .NET para fazer o seguinte:

- Criar uma solicitação de instância spot
- Determinar quando a solicitação de instância spot é atendida
- Cancelar a solicitação de instância spot
- Encerrar as instâncias associadas

As seções a seguir fornecem snippets e outras informações para este exemplo. O código completo do exemplo é mostrado após os snippets e pode ser criado e executado como está.

Tópicos

- Pré-requisitos
- Colete o que você precisa
- Como criar uma solicitação de instância spot
- Determinar o estado da solicitação de instância spot
- Limpar suas solicitações de instância spot
- Limpar as instâncias spot
- Código completo
- Considerações adicionais

Pré-requisitos

Para obter informações sobre os pré-requisitos APIs e, consulte a seção principal (). Trabalhar com o Amazon EC2

Colete o que você precisa

Para criar uma solicitação de instância spot, você precisará de várias coisas.

- O número da instância e o seu tipo. Há alguns tipos de instância a escolher. Para obter mais informações, consulte Tipos de instâncias do Amazon EC2 no Guia do usuário do Amazon EC2. Consulte também Detalhes do tipo de instância e Explorador de tipos de instância.

Para número padrão para este tutorial é o 1.

- A imagem de máquina da Amazon (AMI) que será usada para criar a instância. Para obter informações sobre isso AMIs, consulte Amazon Machine Images (AMIs) no Guia do usuário do Amazon EC2. Em particular, consulte Encontre uma AMI e compartilhe AMIs.

- O preço máximo que você está disposto a pagar por hora por instância. Você pode ver os preços de todos os tipos de instâncias (tanto para instâncias sob demanda quanto para instâncias spot) na página de preços do Amazon EC2. O preço padrão deste tutorial será explicado posteriormente.

- Se você quiser se conectar remotamente a uma instância, a um grupo de segurança com a configuração e os recursos adequados. Isso está descrito em Como trabalhar com grupos de segurança no Amazon EC2 e as informações sobre como coletar o que você precisa e conectar-se a uma instância em Iniciar uma instância do Amazon EC2. Para simplificar, este tutorial usa o grupo de segurança chamado padrão, que todas as contas AWS mais recentes possuem.

Há muitas maneiras de abordar a solicitação de instâncias spot. Estas são estratégias comuns:

- Faça solicitações que certamente custarão menos que os preços sob demanda.
- Solicitar com base no valor da computação resultante.
- Solicitar para adquirir capacidade de computação o mais rápido possível.

As explicações a seguir se referem ao histórico de preços de instâncias spot no Guia do usuário do Amazon EC2.

Reduzir custos abaixo sob demanda

Você tem um job de processamento em lote que modera determinado número de horas ou dias para ser executado. Contudo, você tem flexibilidade em relação ao início e ao fim. Você quer ver se pode o concluído por um custo inferior ao das instâncias sob demanda.

Você examina o histórico de preços spot para tipos de instância usando o console da Amazon EC2 ou a API do Amazon EC2. Depois de analisar o histórico de preços do tipo de instância desejado em determinada zona de disponibilidade, há duas abordagens alternativas para sua solicitação:

- Especificar uma solicitação na extremidade superior do intervalo de preços spot que ainda estão abaixo do preço sob demanda, prevendo que sua solicitação instância spot única seria provavelmente cumprida e executada pelo tempo de computação consecutivo suficiente para concluir o trabalho.
- Especificar uma solicitação na extremidade inferior do intervalo de preço e planejar a combinação de várias instâncias executadas ao longo do tempo por meio de uma requisição persistente. As instâncias seriam executadas por tempo suficiente, ao todo, para concluir o trabalho com um custo total ainda inferior.

Não pagar mais que o valor do resultado

Você tem um job de processamento de dados para ser executado. Você entende o valor dos resultados do trabalho bem o suficiente para saber o quanto valem em termos de custos computação.

Após analisar o histórico de preços spot para seu tipo de instância, escolha o preço de lance no qual o custo do tempo computacional não é mais que o valor dos resultados do job. Você cria uma requisição persistente e a deixa ser executada de forma intermitente, à medida que o preço spot flutua acima ou abaixo da sua solicitação.

Adquirir capacidade de computação rapidamente

Você tem uma necessidade imprevista e de curto prazo para capacidade adicional indisponível pelas instâncias sob demanda. Depois de analisar o histórico de preços spot para seu tipo de instância, você escolhe um preço acima do preço histórico mais alto para aumentar significativamente a probabilidade de sua solicitação ser atendida com rapidez e continuar a computação até a conclusão.

Depois de coletar o que precisa e escolher uma estratégia, você está pronto para solicitar uma Instância Spot. Para este tutorial, o preço máximo da instância spot padrão está definido para ser o mesmo que o preço da instância sob demanda (que é 0,003 USD para este tutorial). Definir o preço desta forma maximiza as chances de que o pedido seja atendido.

Como criar uma solicitação de instância spot

O snippet a seguir mostra como criar uma solicitação de instância spot com os elementos coletados anteriormente.

O exemplo no final deste tópico mostra o snippet em uso.

```
    //
    // Method to create a Spot Instance request
    private static async Task<SpotInstanceRequest> CreateSpotInstanceRequest(
      IAmazonEC2 ec2Client, string amiId, string securityGroupName,
      InstanceType instanceType, string spotPrice, int instanceCount)
    {
      var launchSpecification = new LaunchSpecification{
        ImageId = amiId,
        InstanceType = instanceType
      };
      launchSpecification.SecurityGroups.Add(securityGroupName);
      var request = new RequestSpotInstancesRequest{
        SpotPrice = spotPrice,
        InstanceCount = instanceCount,
        LaunchSpecification = launchSpecification
      };

      RequestSpotInstancesResponse result =
        await ec2Client.RequestSpotInstancesAsync(request);
      return result.SpotInstanceRequests[0];
    }
```

O valor importante retornado desse método é o ID da solicitação da Instância Spot, que está contido no `SpotInstanceRequestId` membro do SpotInstanceRequestobjeto retornado.

> **Note**
>
> Você será cobrado por todas as instâncias spot que forem iniciadas. Para evitar custos desnecessários, certifique-se de cancelar todas as solicitações e encerrar todas as instâncias.

Determinar o estado da solicitação de instância spot

O snippet a seguir mostra como obter informações sobre sua solicitação de instância spot. Você pode usar essas informações para tomar certas decisões em seu código, como decidir continuar esperando o atendimento de uma solicitação de instância spot.

O exemplo no final deste tópico mostra o snippet em uso.

```
    //
```

```
    // Method to get information about a Spot Instance request, including the status,
    // instance ID, etc.
    // It gets the information for a specific request (as opposed to all requests).
    private static async Task<SpotInstanceRequest> GetSpotInstanceRequestInfo(
      IAmazonEC2 ec2Client, string requestId)
    {
      var describeRequest = new DescribeSpotInstanceRequestsRequest();
      describeRequest.SpotInstanceRequestIds.Add(requestId);

      DescribeSpotInstanceRequestsResponse describeResponse =
        await ec2Client.DescribeSpotInstanceRequestsAsync(describeRequest);
      return describeResponse.SpotInstanceRequests[0];
    }
```

O método retorna informações sobre a solicitação da instância spot, como o ID da instância, seu estado e o código de status. Para acessar mais informações sobre os códigos de status das solicitações de Instância spot, consulte Spot request status no Guia do usuário do Amazon EC2.

Limpar suas solicitações de instância spot

Quando você não precisar mais solicitar instâncias spot, é importante cancelar todas as solicitações pendentes para evitar que essas solicitações sejam atendidas novamente. O snippet a seguir mostra como cancelar uma solicitação de instância spot.

O exemplo no final deste tópico mostra o snippet em uso.

```
    //
    // Method to cancel a Spot Instance request
    private static async Task CancelSpotInstanceRequest(
      IAmazonEC2 ec2Client, string requestId)
    {
      var cancelRequest = new CancelSpotInstanceRequestsRequest();
      cancelRequest.SpotInstanceRequestIds.Add(requestId);

      await ec2Client.CancelSpotInstanceRequestsAsync(cancelRequest);
    }
```

Limpar as instâncias spot

Para evitar custos desnecessários, é importante que você encerre todas as instâncias que foram iniciadas a partir de solicitações de instância spot. Apenas cancelar a solicitação de instância spot não encerrará as instâncias, o que significa que você continuará a ser cobrado por elas. O snippet

a seguir mostra como encerrar uma instância depois de obter o identificador da instância para uma instância spot ativa.

O exemplo no final deste tópico mostra o snippet em uso.

```
    //
    // Method to terminate a Spot Instance
    private static async Task TerminateSpotInstance(
      IAmazonEC2 ec2Client, string requestId)
    {
      var describeRequest = new DescribeSpotInstanceRequestsRequest();
      describeRequest.SpotInstanceRequestIds.Add(requestId);

      // Retrieve the Spot Instance request to check for running instances.
      DescribeSpotInstanceRequestsResponse describeResponse =
        await ec2Client.DescribeSpotInstanceRequestsAsync(describeRequest);

      // If there are any running instances, terminate them
      if(   (describeResponse.SpotInstanceRequests[0].Status.Code
              == "request-canceled-and-instance-running")
         || (describeResponse.SpotInstanceRequests[0].State ==
SpotInstanceState.Active))
      {
        TerminateInstancesResponse response =
          await ec2Client.TerminateInstancesAsync(new TerminateInstancesRequest{
            InstanceIds = new List<string>(){
              describeResponse.SpotInstanceRequests[0].InstanceId } });
        foreach (InstanceStateChange item in response.TerminatingInstances)
        {
          Console.WriteLine($"\n  Terminated instance: {item.InstanceId}");
          Console.WriteLine($"  Instance state: {item.CurrentState.Name}\n");
        }
      }
    }
```

Código completo

O exemplo de código a seguir chama os métodos descritos anteriormente para criar e cancelar uma solicitação de instância spot e encerrar uma instância spot.

Referências do SDK

NuGet pacotes:

- AWSSDK.EC2

Elementos de programação:

- Namespace Amazon.EC2

  Classe Amazon EC2 Client

  Classe InstanceType

- Namespace Amazon.EC2.model

  Classe CancelSpotInstanceRequestsRequest

  Classe DescribeSpotInstanceRequestsRequest

  Classe DescribeSpotInstanceRequestsResponse

  Classe InstanceStateChange

  Classe LaunchSpecification

  Classe RequestSpotInstancesRequest

  Classe RequestSpotInstancesResponse

  Classe SpotInstanceRequest

  Classe TerminateInstancesRequest

  Classe TerminateInstancesResponse

O código

```
using System;
using System.Threading;
using System.Threading.Tasks;
using System.Collections.Generic;
using Amazon.EC2;
using Amazon.EC2.Model;
```

```
namespace EC2SpotInstanceRequests
{
  class Program
  {
    static async Task Main(string[] args)
    {
      // Some default values.
      // These could be made into command-line arguments instead.
      var instanceType = InstanceType.T1Micro;
      string securityGroupName = "default";
      string spotPrice = "0.003";
      int instanceCount = 1;

      // Parse the command line arguments
      if((args.Length != 1) || (!args[0].StartsWith("ami-")))
      {
        Console.WriteLine("\nUsage: EC2SpotInstanceRequests ami");
        Console.WriteLine("  ami: the Amazon Machine Image to use for the Spot
 Instances.");
        return;
      }

      // Create the Amazon EC2 client.
      var ec2Client = new AmazonEC2Client();

      // Create the Spot Instance request and record its ID
      Console.WriteLine("\nCreating spot instance request...");
      var req = await CreateSpotInstanceRequest(
        ec2Client, args[0], securityGroupName, instanceType, spotPrice, instanceCount);
      string requestId = req.SpotInstanceRequestId;

      // Wait for an EC2 Spot Instance to become active
      Console.WriteLine(
        $"Waiting for Spot Instance request with ID {requestId} to become active...");
      int wait = 1;
      var start = DateTime.Now;
      while(true)
      {
        Console.Write(".");

        // Get and check the status to see if the request has been fulfilled.
        var requestInfo = await GetSpotInstanceRequestInfo(ec2Client, requestId);
        if(requestInfo.Status.Code == "fulfilled")
        {
```

```
        Console.WriteLine($"\nSpot Instance request {requestId} " +
          $"has been fulfilled by instance {requestInfo.InstanceId}.\n");
        break;
      }

      // Wait a bit and try again, longer each time (1, 2, 4, ...)
      Thread.Sleep(wait);
      wait = wait * 2;
    }

    // Show the user how long it took to fulfill the Spot Instance request.
    TimeSpan span = DateTime.Now.Subtract(start);
    Console.WriteLine($"That took {span.TotalMilliseconds} milliseconds");

    // Perform actions here as needed.
    // For this example, simply wait for the user to hit a key.
    // That gives them a chance to look at the EC2 console to see
    // the running instance if they want to.
    Console.WriteLine("Press any key to start the cleanup...");
    Console.ReadKey(true);

    // Cancel the request.
    // Do this first to make sure that the request can't be re-fulfilled
    // once the Spot Instance has been terminated.
    Console.WriteLine("Canceling Spot Instance request...");
    await CancelSpotInstanceRequest(ec2Client, requestId);

    // Terminate the Spot Instance that's running.
    Console.WriteLine("Terminating the running Spot Instance...");
    await TerminateSpotInstance(ec2Client, requestId);

    Console.WriteLine("Done. Press any key to exit...");
    Console.ReadKey(true);
  }


  //
  // Method to create a Spot Instance request
  private static async Task<SpotInstanceRequest> CreateSpotInstanceRequest(
    IAmazonEC2 ec2Client, string amiId, string securityGroupName,
    InstanceType instanceType, string spotPrice, int instanceCount)
  {
    var launchSpecification = new LaunchSpecification{
      ImageId = amiId,
```

```
        InstanceType = instanceType
      };
      launchSpecification.SecurityGroups.Add(securityGroupName);
      var request = new RequestSpotInstancesRequest{
        SpotPrice = spotPrice,
        InstanceCount = instanceCount,
        LaunchSpecification = launchSpecification
      };

      RequestSpotInstancesResponse result =
        await ec2Client.RequestSpotInstancesAsync(request);
      return result.SpotInstanceRequests[0];
    }


    //
    // Method to get information about a Spot Instance request, including the status,
    // instance ID, etc.
    // It gets the information for a specific request (as opposed to all requests).
    private static async Task<SpotInstanceRequest> GetSpotInstanceRequestInfo(
      IAmazonEC2 ec2Client, string requestId)
    {
      var describeRequest = new DescribeSpotInstanceRequestsRequest();
      describeRequest.SpotInstanceRequestIds.Add(requestId);

      DescribeSpotInstanceRequestsResponse describeResponse =
        await ec2Client.DescribeSpotInstanceRequestsAsync(describeRequest);
      return describeResponse.SpotInstanceRequests[0];
    }


    //
    // Method to cancel a Spot Instance request
    private static async Task CancelSpotInstanceRequest(
      IAmazonEC2 ec2Client, string requestId)
    {
      var cancelRequest = new CancelSpotInstanceRequestsRequest();
      cancelRequest.SpotInstanceRequestIds.Add(requestId);

      await ec2Client.CancelSpotInstanceRequestsAsync(cancelRequest);
    }


    //
```

```
    // Method to terminate a Spot Instance
    private static async Task TerminateSpotInstance(
      IAmazonEC2 ec2Client, string requestId)
    {
      var describeRequest = new DescribeSpotInstanceRequestsRequest();
      describeRequest.SpotInstanceRequestIds.Add(requestId);

      // Retrieve the Spot Instance request to check for running instances.
      DescribeSpotInstanceRequestsResponse describeResponse =
        await ec2Client.DescribeSpotInstanceRequestsAsync(describeRequest);

      // If there are any running instances, terminate them
      if(   (describeResponse.SpotInstanceRequests[0].Status.Code
              == "request-canceled-and-instance-running")
         || (describeResponse.SpotInstanceRequests[0].State ==
 SpotInstanceState.Active))
      {
        TerminateInstancesResponse response =
          await ec2Client.TerminateInstancesAsync(new TerminateInstancesRequest{
            InstanceIds = new List<string>(){
              describeResponse.SpotInstanceRequests[0].InstanceId } });
        foreach (InstanceStateChange item in response.TerminatingInstances)
        {
          Console.WriteLine($"\n  Terminated instance: {item.InstanceId}");
          Console.WriteLine($"  Instance state: {item.CurrentState.Name}\n");
        }
      }
    }
  }
}
```

Considerações adicionais

- Depois de executar o tutorial, é uma boa ideia fazer login no console do Amazon EC2 para verificar se a solicitação da Instância Spot foi cancelada e se a Instância Spot foi encerrada.

Acessando AWS Identity and Access Management (IAM) com o AWS SDK para .NET

O AWS SDK para .NET suporte AWS Identity and Access Management, que é um serviço da web que permite AWS aos clientes gerenciar usuários e permissões de usuário em AWS.

Um usuário AWS Identity and Access Management (IAM) é uma entidade que você cria em AWS. A entidade representa uma pessoa ou aplicativo com AWS quem interage. Para obter mais informações sobre os usuários do IAM, consulte Usuários do IAM e limites do IAM e do STS no Guia do usuário do IAM.

Conceder permissões a um usuário criando uma política do IAM. A política contém um documento de política que lista as ações que um usuário pode executar e os recursos que essas ações podem afetar. Para obter mais informações sobre as políticas do IAM, consulte Permissões e políticas no Guia do usuário do IAM.

> **Warning**
>
> Para evitar riscos de segurança, não use usuários do IAM para autenticação ao desenvolver software com propósito específico ou trabalhar com dados reais. Em vez disso, use federação com um provedor de identidade, como Centro de Identidade do AWS IAM.

APIs

Eles AWS SDK para .NET fornecem APIs para clientes do IAM. Eles APIs permitem que você trabalhe com recursos do IAM, como usuários, funções e chaves de acesso.

Esta seção contém um pequeno número de exemplos que mostram os padrões que você pode seguir ao trabalhar com eles APIs. Para ver o conjunto completo de APIs, consulte a Referência da AWS SDK para .NET API (e vá até “Amazon. IdentityManagement“).

Esta seção também contém um exemplo que mostra como anexar um perfil do IAM às instâncias do Amazon EC2 para facilitar o gerenciamento de credenciais.

O IAM APIs é fornecido pelo AWSSDK. IdentityManagement NuGetpacote.

Pré-requisitos

Antes de começar, verifique se você configurou seu ambiente e configurou seu projeto. Revise também as informações em Uso da SDK.

Tópicos

Tópicos

- Criar políticas gerenciadas pelo IAM a partir do JSON

- Exibir o documento da política gerenciada do IAM
- Conceder acesso utilizando um perfil do IAM

Criar políticas gerenciadas pelo IAM a partir do JSON

Este exemplo mostra como usar o AWS SDK para .NET para criar uma política gerenciada pelo IAM a partir de um determinado documento de política em JSON. O aplicativo cria um objeto cliente do IAM, lê o documento de política de um arquivo e, em seguida, cria a política.

> **Note**
>
> Para ver um exemplo de documento de política em JSON, consulte as considerações adicionais no final deste tópico.

As seções a seguir fornecem snippets desse exemplo. O código completo do exemplo é mostrado depois e pode ser criado e executado como está.

Tópicos

- Crie a política do
- Código completo
- Considerações adicionais

Crie a política do

O trecho a seguir cria uma política gerenciada pelo IAM com o nome e o documento de política fornecidos.

O exemplo no final deste tópico mostra o snippet em uso.

```
    //
    // Method to create an IAM policy from a JSON file
    private static async Task<CreatePolicyResponse> CreateManagedPolicy(
      IAmazonIdentityManagementService iamClient, string policyName, string
jsonFilename)
    {
      return await iamClient.CreatePolicyAsync(new CreatePolicyRequest{
        PolicyName = policyName,
        PolicyDocument = File.ReadAllText(jsonFilename)});
```

```
    }
```

Código completo

Esta seção mostra as referências relevantes e o código completo desse exemplo.

Referências do SDK

NuGet pacotes:

- AWSSDK.IdentityManagement

Elementos de programação:

- Namespace Amazon. IdentityManagement

  Classe AmazonIdentityManagementServiceClient
- Namespace Amazon. IdentityManagement.Modelo

  Classe CreatePolicyRequest

  Classe CreatePolicyResponse

O código

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Threading.Tasks;
using Amazon.IdentityManagement;
using Amazon.IdentityManagement.Model;

namespace IamCreatePolicyFromJson
{
  // = = =
```

```
    {
      // Parse the command line and show help if necessary
      var parsedArgs = CommandLine.Parse(args);
      if((parsedArgs.Count == 0) || (parsedArgs.Count > MaxArgs))
      {
        PrintHelp();
        return;
      }

      // Get the application arguments from the parsed list
      string policyName =
        CommandLine.GetArgument(parsedArgs, null, "-p", "--policy-name");
      string policyFilename =
        CommandLine.GetArgument(parsedArgs, null, "-j", "--json-filename");
      if(   string.IsNullOrEmpty(policyName)
         || (string.IsNullOrEmpty(policyFilename) || !
policyFilename.EndsWith(".json")))
        CommandLine.ErrorExit(
          "\nOne or more of the required arguments is missing or incorrect." +
          "\nRun the command with no arguments to see help.");

      // Create an IAM service client
      var iamClient = new AmazonIdentityManagementServiceClient();

      // Create the new policy
      var response = await CreateManagedPolicy(iamClient, policyName, policyFilename);
      Console.WriteLine($"\nPolicy {response.Policy.PolicyName} has been created.");
      Console.WriteLine($"  Arn: {response.Policy.Arn}");
    }


    //
    // Method to create an IAM policy from a JSON file
    private static async Task<CreatePolicyResponse> CreateManagedPolicy(
      IAmazonIdentityManagementService iamClient, string policyName, string
 jsonFilename)
    {
      return await iamClient.CreatePolicyAsync(new CreatePolicyRequest{
        PolicyName = policyName,
        PolicyDocument = File.ReadAllText(jsonFilename)});
    }


    //
```

```
    // Command-line help
    private static void PrintHelp()
    {
      Console.WriteLine(
        "\nUsage: IamCreatePolicyFromJson -p <policy-name> -j <json-filename>" +
        "\n  -p, --policy-name: The name you want the new policy to have." +
        "\n  -j, --json-filename: The name of the JSON file with the policy
document.");
    }
 }


 // = = =
```

```
          // Check to see if there's a value that goes with this option?
          if((i < args.Length) && (!args[i].StartsWith("-"))) value = args[i++];
          parsedArgs.Add(key, value);
        }

        // If the first argument in this iteration doesn't start with a dash, it's a
value
        else
        {
          parsedArgs.Add("--NoKey" + n.ToString(), args[i++]);
          n++;
        }
      }

      return parsedArgs;
    }

    //
    // Method to get an argument from the parsed command-line arguments
    //
    // Parameters:
    // - parsedArgs: The Dictionary object returned from the Parse() method (shown
above).
    // - defaultValue: The default string to return if the specified key isn't in
parsedArgs.
    // - keys: An array of keys to look for in parsedArgs.
    public static string GetArgument(
      Dictionary<string,string> parsedArgs, string defaultReturn, params string[] keys)
    {
      string retval = null;
      foreach(var key in keys)
        if(parsedArgs.TryGetValue(key, out retval)) break;
      return retval ?? defaultReturn;
    }

    //
    // Method to exit the application with an error.
    public static void ErrorExit(string msg, int code=1)
    {
      Console.WriteLine("\nError");
      Console.WriteLine(msg);
      Environment.Exit(code);
    }
  }
```

```
}
```

Considerações adicionais

- Veja a seguir um exemplo de documento de política que você pode copiar em um arquivo JSON e usar como entrada para esse aplicativo:

  JSON

```
{
  "Version":"2012-10-17",
  "Id"  : "DotnetTutorialPolicy",
  "Statement" : [
    {
      "Sid" : "DotnetTutorialPolicyS3",
      "Effect" : "Allow",
      "Action" : [
        "s3:Get*",
        "s3:List*"
      ],
      "Resource" : "*"
    },
    {
      "Sid" : "DotnetTutorialPolicyPolly",
      "Effect": "Allow",
      "Action": [
        "polly:DescribeVoices",
        "polly:SynthesizeSpeech"
      ],
      "Resource": "*"
    }
  ]
}
```

- Você pode verificar se a política foi criada consultando o console do IAM. Usar a lista suspensa Filtrar políticas para selecionar Gerenciado pelo cliente. Excluir uma política quando não precisar mais dela.

- Para obter mais informações sobre a criação de políticas, consulte Criação de políticas do IAM e a referência da política JSON no Guia do usuário do IAM

Exibir o documento da política gerenciada do IAM

Este exemplo mostra como usar o AWS SDK para .NET para exibir um documento de política. O aplicativo cria um objeto cliente do IAM, encontra a versão padrão da política gerenciada do IAM em questão e, em seguida, exibe o documento da política em formato JSON.

As seções a seguir fornecem snippets desse exemplo. O código completo do exemplo é mostrado depois e pode ser criado e executado como está.

Tópicos

- Encontrar a versão padrão
- Exibir o documento da política
- Código completo

Encontrar a versão padrão

Os snippet a seguir encontram a versão padrão da política do IAM.

O exemplo no final deste tópico mostra o snippet em uso.

```
//
// Method to determine the default version of an IAM policy
// Returns a string with the version
private static async Task<string> GetDefaultVersion(
  IAmazonIdentityManagementService iamClient, string policyArn)
{
  // Retrieve all the versions of this policy
  string defaultVersion = string.Empty;
  ListPolicyVersionsResponse reponseVersions =
    await iamClient.ListPolicyVersionsAsync(new ListPolicyVersionsRequest{
      PolicyArn = policyArn});

  // Find the default version
  foreach(PolicyVersion version in reponseVersions.Versions)
  {
    if(version.IsDefaultVersion)
    {
      defaultVersion = version.VersionId;
      break;
    }
  }
```

```
      return defaultVersion;
    }
```

Exibir o documento da política

O snippet a seguir exibe o documento de política JSON da política do IAM fornecida.

O exemplo no final deste tópico mostra o snippet em uso.

```
    //
    // Method to retrieve and display the policy document of an IAM policy
    private static async Task ShowPolicyDocument(
      IAmazonIdentityManagementService iamClient, string policyArn, string
defaultVersion)
    {
      // Retrieve the policy document of the default version
      GetPolicyVersionResponse responsePolicy =
        await iamClient.GetPolicyVersionAsync(new GetPolicyVersionRequest{
          PolicyArn = policyArn,
          VersionId = defaultVersion});

      // Display the policy document (in JSON)
      Console.WriteLine($"Version {defaultVersion} of the policy (in JSON format):");
      Console.WriteLine(
        $"{HttpUtility.UrlDecode(responsePolicy.PolicyVersion.Document)}");
    }
```

Código completo

Esta seção mostra as referências relevantes e o código completo desse exemplo.

Referências do SDK

NuGet pacotes:

- AWSSDK.IdentityManagement

Elementos de programação:

- Namespace Amazon. IdentityManagement

  Classe AmazonIdentityManagementServiceClient

- Namespace Amazon. IdentityManagement.Modelo

  Classe GetPolicyVersionRequest

  Classe GetPolicyVersionResponse

  Classe ListPolicyVersionsRequest

  Classe ListPolicyVersionsResponse

  Classe PolicyVersion

O código

```
using System;
using System.Web;
using System.Threading.Tasks;
using Amazon.IdentityManagement;
using Amazon.IdentityManagement.Model;

namespace IamDisplayPolicyJson
{
  class Program
  {
    static async Task Main(string[] args)
    {
      // Parse the command line and show help if necessary
      if(args.Length != 1)
      {
        Console.WriteLine("\nUsage: IamDisplayPolicyJson policy-arn");
        Console.WriteLine("   policy-arn: The ARN of the policy to retrieve.");
        return;
      }
      if(!args[0].StartsWith("arn:"))
      {
        Console.WriteLine("\nCould not find policy ARN in the command-line
 arguments:");
        Console.WriteLine($"{args[0]}");
        return;
      }

      // Create an IAM service client
      var iamClient = new AmazonIdentityManagementServiceClient();
```

```
      // Retrieve and display the policy document of the given policy
      string defaultVersion = await GetDefaultVersion(iamClient, args[0]);
      if(string.IsNullOrEmpty(defaultVersion))
        Console.WriteLine($"Could not find the default version for policy {args[0]}.");
      else
        await ShowPolicyDocument(iamClient, args[0], defaultVersion);
    }


    //
    // Method to determine the default version of an IAM policy
    // Returns a string with the version
    private static async Task<string> GetDefaultVersion(
      IAmazonIdentityManagementService iamClient, string policyArn)
    {
      // Retrieve all the versions of this policy
      string defaultVersion = string.Empty;
      ListPolicyVersionsResponse reponseVersions =
        await iamClient.ListPolicyVersionsAsync(new ListPolicyVersionsRequest{
          PolicyArn = policyArn});

      // Find the default version
      foreach(PolicyVersion version in reponseVersions.Versions)
      {
        if(version.IsDefaultVersion)
        {
          defaultVersion = version.VersionId;
          break;
        }
      }

      return defaultVersion;
    }


    //
    // Method to retrieve and display the policy document of an IAM policy
    private static async Task ShowPolicyDocument(
      IAmazonIdentityManagementService iamClient, string policyArn, string
  defaultVersion)
    {
      // Retrieve the policy document of the default version
      GetPolicyVersionResponse responsePolicy =
```

```
        await iamClient.GetPolicyVersionAsync(new GetPolicyVersionRequest{
          PolicyArn = policyArn,
          VersionId = defaultVersion});

      // Display the policy document (in JSON)
      Console.WriteLine($"Version {defaultVersion} of the policy (in JSON format):");
      Console.WriteLine(
        $"{HttpUtility.UrlDecode(responsePolicy.PolicyVersion.Document)}");
    }
  }
}
```

Conceder acesso utilizando um perfil do IAM

Este tutorial mostra como usar o para AWS SDK para .NET habilitar funções do IAM em instâncias do Amazon EC2.

Visão geral do

Todas as solicitações AWS devem ser assinadas criptograficamente usando credenciais emitidas por. AWS Portanto, é necessária uma estratégia para gerenciar credenciais para aplicações executadas nas instâncias do Amazon EC2. É necessário distribuir, armazenar e girar essas credenciais com segurança, mas também mantendo-as acessíveis à aplicação.

Com os perfis do IAM, você pode gerenciar essas credenciais com eficiência. Você cria um perfil do IAM e a configura com as permissões que uma aplicação exige e, em seguida, anexa esse perfil a uma instância do EC2. Para ler mais sobre os benefícios de usar perfis do IAM, consulte Perfis do IAM para o Amazon EC2 no Guia do usuário do Amazon EC2. Consulte também as informações sobre perfis do IAM no Guia do usuário do IAM.

Para um aplicativo criado usando o AWS SDK para .NET, quando o aplicativo constrói um objeto cliente para um AWS serviço, o objeto pesquisa credenciais de várias fontes potenciais. A ordem da pesquisa é mostrada em Resolução de perfil e credenciais.

Se o objeto cliente não encontrar as credenciais em nenhuma outra fonte, ele recuperará as credenciais temporárias com as mesmas permissões que as associadas ao perfil do IAM pelos metadados da instância EC2. Essas credenciais são usadas para fazer chamadas a AWS partir do objeto cliente.

Sobre este tutorial

Ao seguir este tutorial, você usa a AWS SDK para .NET (e outras ferramentas) para iniciar uma instância do Amazon EC2 com uma função do IAM anexada e, em seguida, vê um aplicativo na instância usando as permissões da função do IAM.

Tópicos

- Criar uma amostra da aplicação no Amazon S3
- Criar um perfil do IAM
- Iniciar uma instância do EC2 e anexar o perfil do IAM
- Conectar-se à instância EC2
- Executar o exemplo da aplicação na instância do EC2
- Limpeza

Criar uma amostra da aplicação no Amazon S3

Esta aplicação de exemplo recupera um objeto do Amazon S3. Para executar a aplicação, você precisa:

- Um bucket do Amazon S3 que contém um arquivo de texto.
- AWS credenciais em sua máquina de desenvolvimento que permitem acessar o bucket.

Para ter informações sobre como criar um bucket do Amazon S3 e fazer upload de um objeto, consulte o Guia do usuário do Amazon Simple Storage Service. Para obter informações sobre AWS credenciais, consulteAutenticando o com AWS SDK para .NET AWS.

Criar um projeto .NET Core com o seguinte código. Em seguida, teste aplicação em sua máquina de desenvolvimento.

> **Note**
>
> Em sua máquina de desenvolvimento, o domínio .NET Core Runtime está instalado, o que permite que você execute a aplicação sem publicá-la. Ao criar uma instância do EC2 mais adiante neste tutorial, é possível instalar o. NET Core Runtime na instância. Isso proporciona uma experiência semelhante e uma transferência de arquivos menor.

No entanto, é possível optar por não instalar o.NET Core Runtime na instância. Se você escolher esse curso de ação, deverá publicar a aplicação para que todas as dependências sejam incluídas ao transferi-la para a instância.

Referências do SDK

NuGet pacotes:

- AWSSDK.S3

Elementos de programação:

- Namespace Amazon.S3

  Classe AmazonS3Client
- Namespace Amazon.S3.Model

  Classe GetObjectResponse

O código

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Model;

namespace S3GetTextItem
{
  // = = =
```

```
    {
      PrintHelp();
      return;
    }

    // Get the application arguments from the parsed list
    string bucket =
      CommandLine.GetArgument(parsedArgs, null, "-b", "--bucket-name");
    string item =
      CommandLine.GetArgument(parsedArgs, null, "-t", "--text-object");
    string outFile =
      CommandLine.GetArgument(parsedArgs, null, "-o", "--output-filename");
    if(   string.IsNullOrEmpty(bucket)
       || string.IsNullOrEmpty(item)
       || string.IsNullOrEmpty(outFile))
      CommandLine.ErrorExit(
        "\nOne or more of the required arguments is missing or incorrect." +
        "\nRun the command with no arguments to see help.");

    // Create the S3 client object and get the file object from the bucket.
    var response = await GetObject(new AmazonS3Client(), bucket, item);

    // Write the contents of the file object to the given output file.
    var reader = new StreamReader(response.ResponseStream);
    string contents = reader.ReadToEnd();
    using (var s = new FileStream(outFile, FileMode.Create))
    using (var writer = new StreamWriter(s))
      writer.WriteLine(contents);
  }


  //
  // Method to get an object from an S3 bucket.
  private static async Task<GetObjectResponse> GetObject(
    IAmazonS3 s3Client, string bucket, string item)
  {
      Console.WriteLine($"Retrieving {item} from bucket {bucket}.");
      return await s3Client.GetObjectAsync(bucket, item);
  }


  //
  // Command-line help
  private static void PrintHelp()
```

```
    {
      Console.WriteLine(
        "\nUsage: S3GetTextItem -b <bucket-name> -t <text-object> -o <output-filename>"
+
        "\n  -b, --bucket-name: The name of the S3 bucket." +
        "\n  -t, --text-object: The name of the text object in the bucket." +
        "\n  -o, --output-filename: The name of the file to write the text to.");
    }
  }


  // = = =
```

```
          if((i < args.Length) && (!args[i].StartsWith("-"))) value = args[i++];
          parsedArgs.Add(key, value);
        }

        // If the first argument in this iteration doesn't start with a dash, it's a
value
        else
        {
          parsedArgs.Add("--NoKey" + n.ToString(), args[i++]);
          n++;
        }
      }

      return parsedArgs;
    }

    //
    // Method to get an argument from the parsed command-line arguments
    //
    // Parameters:
    // - parsedArgs: The Dictionary object returned from the Parse() method (shown
above).
    // - defaultValue: The default string to return if the specified key isn't in
parsedArgs.
    // - keys: An array of keys to look for in parsedArgs.
    public static string GetArgument(
      Dictionary<string,string> parsedArgs, string defaultReturn, params string[] keys)
    {
      string retval = null;
      foreach(var key in keys)
        if(parsedArgs.TryGetValue(key, out retval)) break;
      return retval ?? defaultReturn;
    }

    //
    // Method to exit the application with an error.
    public static void ErrorExit(string msg, int code=1)
    {
      Console.WriteLine("\nError");
      Console.WriteLine(msg);
      Environment.Exit(code);
    }
  }
```

```
}
```

Se quiser, você pode remover temporariamente as credenciais usadas na máquina de desenvolvimento para ver como a aplicação responde. (Mas não se esqueça de restaurar as credenciais quando concluir).

Criar um perfil do IAM

Criar um perfil do IAM com as permissões apropriadas para acessar o Amazon S3.

1. Abra o console do IAM.
2. No painel de navegação, escolha Perfis e depois escolha Criar perfil.
3. Selecionar serviço da AWS , escolher EC2 e depois Avançar: Permissões.
4. Em Anexar políticas de permissões, encontre e selecione ReadOnlyAccessAmazonS3. Revise a política, se quiser, e escolha Avançar: Tags.
5. Adicione tags, se quiser, e escolha Avançar: Revisão.
6. Digite um nome e uma descrição para a função e, em seguida, escolha Criar função. Lembre-se desse nome, pois você precisará dele quando executar sua instância do EC2.

Iniciar uma instância do EC2 e anexar o perfil do IAM

Iniciar uma instância do EC2 com o perfil do IAM que foi criada. Você pode fazer isso das seguintes maneiras.

- Usar o console do EC2

  Para executar uma instância usando o console do EC2, consulte Iniciar uma instância usando o novo assistente de inicialização de instâncias no Guia do usuário do Amazon EC2.

  Ao examinar a página de lançamento, você deve pelo menos expandir o painel Detalhes avançados para poder especificar o perfil do IAM que você criou anteriormente no perfil da instância do IAM.

- Usando o AWS SDK para .NET

  Para obter informações sobre isso, consulte Iniciar uma instância do Amazon EC2, incluindo o Considerações adicionais próximo ao final desse tópico.

Para iniciar uma instância do EC2 com um perfil do IAM anexado, uma configuração de um usuário do IAM deve incluir determinadas permissões. Para acessar mais informações sobre as permissões necessárias, consulte Conceder permissão a um usuário para passar um perfil do IAM para uma instância no Guia do usuário do Amazon EC2.

Conectar-se à instância EC2

Conectar à instância do EC2 para que você possa transferir a amostra do aplicativo e, em seguida, executar o aplicativo. Você precisará do arquivo que contém a parte privada do par de chaves usado para executar a instância, ou seja, o arquivo PEM.

Para acessar informações sobre como se conectar à instância, consulte Conectar-se à sua instância do Linux ou Conectar-se à instância do Windows no Guia do usuário do Amazon EC2. Ao conectar-se, faça isso de maneira que possa transferir arquivos da sua máquina de desenvolvimento para sua instância.

Se estiver usando o Visual Studio no Windows, também poderá conectar-se à instância usando o Toolkit for Visual Studio. Para obter mais informações, consulte Conectando-se a uma instância do Amazon EC2 no Guia do AWS Toolkit for Visual Studio usuário.

Executar o exemplo da aplicação na instância do EC2

1. Copiar as aplicações do aplicativo da sua unidade local para sua instância.

   Os arquivos transferidos dependem de como a aplicação foi criada e se sua instância tem o.NET Core Runtime instalado. Para acessar informações sobre como transferir arquivos para sua instância, consulte Conecte-se à sua instância Linux (consulte a subseção apropriada) ou Transferir arquivos para instâncias do Windows no Guia do usuário do Amazon EC2.
2. Iniciar a aplicação e verificar se é executada com os mesmos resultados da sua máquina de desenvolvimento.
3. Verifique se a aplicação usa as credenciais fornecidas pelo perfil do IAM.

   a. Abra o console do Amazon EC2.

   b. Selecione a instância e desanexe o perfil do IAM por meio de Ações, Configurações da instância e Anexar/substituir perfil do IAM.

   c. Execute o aplicativo novamente e veja se retorna um erro de autorização.

Limpeza

Ao terminar este tutorial, se não desejar mais a instância do EC2 criada, certifique-se de encerrar a instância para evitar custos indesejados. Isso poderá ser feito no console do Amazon EC2 ou programaticamente, conforme descrito em Encerramento de uma instância Amazon EC2. Se desejar, você também pode excluir outros recursos que tenha criado para este tutorial. Isso pode incluir um perfil do IAM, um par de chaves do EC2 e um arquivo PEM, um grupo de segurança etc.

Usar o armazenamento para internet Amazon Simple Storage Service

O AWS SDK para .NET suporta o Amazon S3, que é armazenamento para a Internet. Ele foi projetado para facilitar a computação de escala na web para os desenvolvedores.

APIs

AWS SDK para .NET Ele fornece APIs para clientes do Amazon S3. Eles APIs permitem que você trabalhe com recursos do Amazon S3, como buckets e itens. Para ver o conjunto completo de APIs para o Amazon S3, veja o seguinte:

- AWS SDK para .NET Referência da API (e vá até “Amazon.S3").
- Documentação Amazon.Extensions.S3.Encryption

O Amazon S3 APIs é fornecido pelos seguintes pacotes: NuGet

- AWSSDK.S3
- Amazon.Extensions.S3.Encryption

Pré-requisitos

Antes de começar, verifique se você configurou seu ambiente e configurou seu projeto. Revise também as informações em Uso da SDK.

Exemplos neste documento

Os tópicos a seguir neste documento mostram como usar o AWS SDK para .NET para trabalhar com o Amazon S3.

- Usar chaves KMS para a criptografia do S3

Exemplos em outros documentos

Os links a seguir para o Amazon S3 Developer Guide fornecem exemplos adicionais de como usar o AWS SDK para .NET para trabalhar com o Amazon S3.

> **Note**
>
> Embora esses exemplos e considerações adicionais de programação tenham sido criados para a versão 3 do AWS SDK para .NET uso do.NET Framework, eles também são viáveis para versões posteriores do AWS SDK para .NET uso do.NET Core. Às vezes, pequenos ajustes no código são necessários.

Exemplos de programação do Amazon S3

- Gerenciando ACLs
- Criação de um bucket
- Upload de um objeto
- Upload em várias partes com a API de alto nível (Amazon.S3.Transfer). TransferUtility)
- Multipart Upload com a API de baixo nível
- Listar objetos
- Chaves de listagem
- Obter um objeto
- Copiar um objeto
- Copiar um objeto com a API do Multipart Upload
- Exclusão de um objeto
- Excluir de vários objetos
- Restaurar um objeto
- Configurar um bucket para notificações
- Gerenciar o ciclo de vida de um objeto
- Gerenciar um URL de objeto pré-assinado
- Gerenciamento de sites
- Ativação do compartilhamento de recursos de origem cruzada (CORS)

Considerações adicionais sobre programação

- Usar o AWS SDK para .NET para o Amazon S3 Programming
- Fazer solicitações usando as credenciais temporárias do usuário do IAM
- Fazer solicitações usando as credenciais temporárias de usuário federado
- Especificar a criptografia no lado do servidor
- Especificar a criptografia do lado do servidor com chaves de criptografia fornecidas pelo cliente

Usando AWS KMS chaves para criptografia do Amazon S3 no AWS SDK para .NET

Este exemplo mostra como usar AWS Key Management Service chaves para criptografar objetos do Amazon S3. O aplicativo cria uma chave mestra do cliente (CMK) e a usa para criar um objeto AmazonS3 EncryptionClient V2 para criptografia do lado do cliente. O aplicativo usa esse cliente para criar um objeto criptografado a partir de um determinado arquivo de texto em um bucket existente do Amazon S3. Em seguida, ele descriptografa o objeto e exibe seu conteúdo.

> **Warning**
>
> Uma classe similar chamada `AmazonS3EncryptionClient` está obsoleta e é menos segura do que a classe `AmazonS3EncryptionClientV2`. Para migrar o código existente que usa `AmazonS3EncryptionClient`, consulte Migração do cliente de criptografia S3 (V1 para V2).

Tópicos

- Criar materiais de criptografia
- Criar e criptografar um objeto do Amazon S3
- Código completo
- Considerações adicionais

Criar materiais de criptografia

O trecho a seguir cria um objeto `EncryptionMaterials` que contém uma ID de chave KMS.

O exemplo no final deste tópico mostra o snippet em uso.

```
    // Create a customer master key (CMK) and store the result
    CreateKeyResponse createKeyResponse =
      await new AmazonKeyManagementServiceClient().CreateKeyAsync(new
CreateKeyRequest());
    var kmsEncryptionContext = new Dictionary<string, string>();
    var kmsEncryptionMaterials = new EncryptionMaterialsV2(
      createKeyResponse.KeyMetadata.KeyId, KmsType.KmsContext, kmsEncryptionContext);
```

Criar e criptografar um objeto do Amazon S3

O snippet a seguir cria um objeto `AmazonS3EncryptionClientV2` que usa os materiais de criptografia criados anteriormente. Em seguida, ele usa o cliente para criar e criptografar um novo objeto do Amazon S3.

O exemplo no final deste tópico mostra o snippet em uso.

```
  //
  // Method to create and encrypt an object in an S3 bucket
  static async Task<GetObjectResponse> CreateAndRetrieveObjectAsync(
    EncryptionMaterialsV2 materials, string bucketName,
    string fileName, string itemName)
  {
    // CryptoStorageMode.ObjectMetadata is required for KMS EncryptionMaterials
    var config = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2AndLegacy)
    {
      StorageMode = CryptoStorageMode.ObjectMetadata
    };
    var s3EncClient = new AmazonS3EncryptionClientV2(config, materials);

    // Create, encrypt, and put the object
    await s3EncClient.PutObjectAsync(new PutObjectRequest
    {
      BucketName = bucketName,
      Key = itemName,
      ContentBody = File.ReadAllText(fileName)
    });

    // Get, decrypt, and return the object
    return await s3EncClient.GetObjectAsync(new GetObjectRequest
    {
      BucketName = bucketName,
      Key = itemName
    });
```

```
    }
```

Código completo

Esta seção mostra as referências relevantes e o código completo desse exemplo.

Referências do SDK

NuGet pacotes:

- Amazon.Extensions.S3.Encryption

Elementos de programação:

- Namespace Amazon.Extensions.S3.Encryption

  Classe EncryptionClientAmazonS3 V2

  Classe CryptoConfigurationAmazonS3 V2

  Classe CryptoStorageMode

  Classe EncryptionMaterialsV2
- Namespace Amazon.Extensions.S3.Encryption.Primitives

  Classe KmsType
- Namespace Amazon.S3.Model

  Classe GetObjectRequest

  Classe GetObjectResponse

  Classe PutObjectRequest
- Namespace Amazon. KeyManagementService

  Classe AmazonKeyManagementServiceClient
- Namespace Amazon. KeyManagementService.Modelo

  Classe CreateKeyRequest

  Classe CreateKeyResponse

O código

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Threading.Tasks;
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;
using Amazon.S3.Model;
using Amazon.KeyManagementService;
using Amazon.KeyManagementService.Model;

namespace KmsS3Encryption
{
  // = = =
```

```
      if(string.IsNullOrEmpty(itemName))
        itemName = Path.GetFileName(fileName);

      // Create a customer master key (CMK) and store the result
      CreateKeyResponse createKeyResponse =
        await new AmazonKeyManagementServiceClient().CreateKeyAsync(new
CreateKeyRequest());
      var kmsEncryptionContext = new Dictionary<string, string>();
      var kmsEncryptionMaterials = new EncryptionMaterialsV2(
        createKeyResponse.KeyMetadata.KeyId, KmsType.KmsContext, kmsEncryptionContext);

      // Create the object in the bucket, then display the content of the object
      var putObjectResponse =
        await CreateAndRetrieveObjectAsync(kmsEncryptionMaterials, bucketName,
fileName, itemName);
      Stream stream = putObjectResponse.ResponseStream;
      StreamReader reader = new StreamReader(stream);
      Console.WriteLine(reader.ReadToEnd());
    }


    //
    // Method to create and encrypt an object in an S3 bucket
    static async Task<GetObjectResponse> CreateAndRetrieveObjectAsync(
      EncryptionMaterialsV2 materials, string bucketName,
      string fileName, string itemName)
    {
      // CryptoStorageMode.ObjectMetadata is required for KMS EncryptionMaterials
      var config = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2AndLegacy)
      {
        StorageMode = CryptoStorageMode.ObjectMetadata
      };
      var s3EncClient = new AmazonS3EncryptionClientV2(config, materials);

      // Create, encrypt, and put the object
      await s3EncClient.PutObjectAsync(new PutObjectRequest
      {
        BucketName = bucketName,
        Key = itemName,
        ContentBody = File.ReadAllText(fileName)
      });

      // Get, decrypt, and return the object
      return await s3EncClient.GetObjectAsync(new GetObjectRequest
```

```
      {
        BucketName = bucketName,
        Key = itemName
      });
    }


    //
    // Command-line help
    private static void PrintHelp()
    {
      Console.WriteLine(
        "\nUsage: KmsS3Encryption -b <bucket-name> -f <file-name> [-i <item-name>]" +
        "\n  -b, --bucket-name: The name of an existing S3 bucket." +
        "\n  -f, --file-name: The name of a text file with content to encrypt and store
in S3." +
        "\n  -i, --item-name: The name you want to use for the item." +
        "\n      If item-name isn't given, file-name will be used.");
    }

 }

 // = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = =
 // Class that represents a command line on the console or terminal.
 // (This is the same for all examples. When you have seen it once, you can ignore
it.)
 static class CommandLine
 {
    //
    // Method to parse a command line of the form: "--key value" or "-k value".
    //
    // Parameters:
    // - args: The command-line arguments passed into the application by the system.
    //
    // Returns:
    // A Dictionary with string Keys and Values.
    //
    // If a key is found without a matching value, Dictionary.Value is set to the key
    //  (including the dashes).
    // If a value is found without a matching key, Dictionary.Key is set to "--NoKeyN",
    //  where "N" represents sequential numbers.
    public static Dictionary<string,string> Parse(string[] args)
    {
```

```
      var parsedArgs = new Dictionary<string,string>();
      int i = 0, n = 0;
      while(i < args.Length)
      {
        // If the first argument in this iteration starts with a dash it's an option.
        if(args[i].StartsWith("-"))
        {
          var key = args[i++];
          var value = key;

          // Check to see if there's a value that goes with this option?
          if((i < args.Length) && (!args[i].StartsWith("-"))) value = args[i++];
          parsedArgs.Add(key, value);
        }

        // If the first argument in this iteration doesn't start with a dash, it's a
value
        else
        {
          parsedArgs.Add("--NoKey" + n.ToString(), args[i++]);
          n++;
        }
      }

      return parsedArgs;
    }

    //
    // Method to get an argument from the parsed command-line arguments
    //
    // Parameters:
    // - parsedArgs: The Dictionary object returned from the Parse() method (shown
above).
    // - defaultValue: The default string to return if the specified key isn't in
parsedArgs.
    // - keys: An array of keys to look for in parsedArgs.
    public static string GetArgument(
      Dictionary<string,string> parsedArgs, string defaultReturn, params string[] keys)
    {
      string retval = null;
      foreach(var key in keys)
        if(parsedArgs.TryGetValue(key, out retval)) break;
      return retval ?? defaultReturn;
    }
```

```
    //
    // Method to exit the application with an error.
    public static void ErrorExit(string msg, int code=1)
    {
      Console.WriteLine("\nError");
      Console.WriteLine(msg);
      Environment.Exit(code);
    }
  }

}
```

Considerações adicionais

- Você pode verificar os resultados desse exemplo. Para fazer isso, acesse o console do Amazon S3 e abra o bucket que você forneceu à aplicação. Em seguida, encontre o novo objeto, baixe-o e abra-o em um editor de texto.

- A classe AmazonS3 EncryptionClient V2 implementa a mesma interface da classe padrão. `AmazonS3Client` Isso facilita a portabilidade do código para a classe `AmazonS3EncryptionClientV2` de modo que a criptografia e a descriptografia ocorram automaticamente e de maneira transparente no cliente.

- Uma vantagem de usar uma AWS KMS chave como chave mestra é que você não precisa armazenar e gerenciar suas próprias chaves mestras; isso é feito por AWS. Uma segunda vantagem é que a `AmazonS3EncryptionClientV2` classe do AWS SDK para .NET é interoperável com a `AmazonS3EncryptionClientV2` classe do. AWS SDK para Java Isso significa que você pode criptografar com o AWS SDK para Java e descriptografar com o AWS SDK para .NET e vice-versa.

  > **Note**
  >
  > A `AmazonS3EncryptionClientV2` classe do AWS SDK para .NET suporta chaves mestras do KMS somente quando executada no modo de metadados. O modo do arquivo de instruções da `AmazonS3EncryptionClientV2` classe do AWS SDK para .NET é incompatível com a `AmazonS3EncryptionClientV2` classe do AWS SDK para Java.

- Para obter mais informações sobre a criptografia do lado do cliente com a `AmazonS3EncryptionClientV2` classe e como a criptografia de envelope funciona, consulte Criptografia de dados do lado do cliente com o Amazon AWS SDK para .NET S3.

Enviar notificações da nuvem usando o Amazon Simple Notification Service

> **Note**
>
> As informações neste tópico são específicas para projetos baseados no.NET Framework e na AWS SDK para .NET versão 3.3 e anteriores.

O é AWS SDK para .NET compatível com o Amazon Simple Notification Service (Amazon SNS), que é um serviço web que permite que aplicativos, usuários finais e dispositivos enviem notificações instantaneamente da nuvem. Para obter mais informações, consulte Amazon SNS.

Listar seus tópicos do Amazon SNS

O exemplo a seguir mostra como listar os tópicos do Amazon SNS, as assinaturas de cada tópico e os atributos de cada tópico. Este exemplo usa o padrão AmazonSimpleNotificationServiceClient.

```
// using Amazon.SimpleNotificationService;
// using Amazon.SimpleNotificationService.Model;

var client = new AmazonSimpleNotificationServiceClient();
var request = new ListTopicsRequest();
var response = new ListTopicsResponse();

do
{
  response = client.ListTopics(request);

  foreach (var topic in response.Topics)
  {
    Console.WriteLine("Topic: {0}", topic.TopicArn);

    var subs = client.ListSubscriptionsByTopic(
      new ListSubscriptionsByTopicRequest
      {
        TopicArn = topic.TopicArn
      });
```

```
    var ss = subs.Subscriptions;

    if (ss.Any())
    {
      Console.WriteLine("  Subscriptions:");

      foreach (var sub in ss)
      {
        Console.WriteLine("    {0}", sub.SubscriptionArn);
      }
    }

    var attrs = client.GetTopicAttributes(
      new GetTopicAttributesRequest
      {
        TopicArn = topic.TopicArn
      }).Attributes;

    if (attrs.Any())
    {
      Console.WriteLine("  Attributes:");

      foreach (var attr in attrs)
      {
        Console.WriteLine("    {0} = {1}", attr.Key, attr.Value);
      }
    }

    Console.WriteLine();
  }

  request.NextToken = response.NextToken;

} while (!string.IsNullOrEmpty(response.NextToken));
```

Enviar uma mensagem para um tópico do Amazon SNS

O exemplo a seguir mostra como enviar uma mensagem para um tópico do SNS. O exemplo usa um argumento, o ARN do tópico do Amazon SNS.

```
using System;
using System.Linq;
```

```
using System.Threading.Tasks;

using Amazon;
using Amazon.SimpleNotificationService;
using Amazon.SimpleNotificationService.Model;

namespace SnsSendMessage
{
    class Program
    {
        static void Main(string[] args)
        {
            /* Topic ARNs must be in the correct format:
             *   arn:aws:sns:REGION:ACCOUNT_ID:NAME
             *
             *  where:
             *  REGION     is the region in which the topic is created, such as us-
west-2
             *  ACCOUNT_ID is your (typically) 12-character account ID
             *  NAME       is the name of the topic
             */
            string topicArn = args[0];
            string message = "Hello at " + DateTime.Now.ToShortTimeString();

            var client = new AmazonSimpleNotificationServiceClient(region:
 Amazon.RegionEndpoint.USWest2);

            var request = new PublishRequest
            {
                Message = message,
                TopicArn = topicArn
            };

            try
            {
                var response = client.Publish(request);

                Console.WriteLine("Message sent to topic:");
                Console.WriteLine(message);
            }
            catch (Exception ex)
            {
                Console.WriteLine("Caught exception publishing request:");
                Console.WriteLine(ex.Message);
```

```
            }
        }
    }
}
```

Veja o exemplo completo, incluindo informações sobre como criar e executar o exemplo na linha de comando, em GitHub.

Enviar uma mensagem SMS para um número de telefone

O exemplo a seguir mostra como enviar uma mensagem SMS para um número de telefone. O exemplo usa um argumento, o número de telefone, que deve estar em um dos dois formatos descritos nos comentários.

```
using System;
using System.Linq;
using System.Threading.Tasks;
using Amazon;
using Amazon.SimpleNotificationService;
using Amazon.SimpleNotificationService.Model;

namespace SnsPublish
{
    class Program
    {
        static void Main(string[] args)
        {
            // US phone numbers must be in the correct format:
            // +1 (nnn) nnn-nnnn OR +1nnnnnnnnnn
            string number = args[0];
            string message = "Hello at " + DateTime.Now.ToShortTimeString();

            var client = new AmazonSimpleNotificationServiceClient(region:
 Amazon.RegionEndpoint.USWest2);
            var request = new PublishRequest
            {
                Message = message,
                PhoneNumber = number
            };

            try
            {
                var response = client.Publish(request);
```

```
                Console.WriteLine("Message sent to " + number + ":");
                Console.WriteLine(message);
            }
            catch (Exception ex)
            {
                Console.WriteLine("Caught exception publishing request:");
                Console.WriteLine(ex.Message);
            }
        }
    }
}
```

Veja o exemplo completo, incluindo informações sobre como criar e executar o exemplo na linha de comando, em GitHub.

Enviar mensagens usando o Amazon SQS

O AWS SDK para .NET suporta o Amazon Simple Queue Service (Amazon SQS), que é um serviço de enfileiramento de mensagens que manipula mensagens ou fluxos de trabalho entre componentes em um sistema.

As filas do Amazon SQS fornecem um mecanismo que permite enviar, armazenar e receber mensagens entre componentes de software, como microsserviços, sistemas distribuídos e aplicações sem servidor. Isso permite desacoplar esses componentes e você fica da necessidade de projetar e operar seu próprio sistema de mensagens. Para ter informações sobre como as filas e mensagens funcionam no Amazon SQS, consulte os tutoriais do Amazon SQS e Arquitetura básica do Amazon SQS no Guia do desenvolvedor do Amazon Simple Queue Service.

> **Important**
>
> Devido à natureza distribuída da fila, o Amazon SQS não garante que as mensagens serão recebidas na ordem exata de envio. Se for necessário manter a ordem de mensagens, use uma fila FIFO do Amazon SQS.

APIs

AWS SDK para .NET Ele fornece APIs para clientes do Amazon SQS. Eles APIs permitem que você trabalhe com recursos do Amazon SQS, como filas e mensagens. Esta seção contém um pequeno

número de exemplos que mostram os padrões que você pode seguir ao trabalhar com eles APIs. Para ver o conjunto completo de APIs, consulte a Referência da AWS SDK para .NET API (e vá até “Amazon.sqs”).

O Amazon SQS é fornecido pelo APIs AWSSDK NuGet pacote.SQS.

Pré-requisitos

Antes de começar, verifique se você configurou seu ambiente e configurou seu projeto. Revise também as informações em Uso da SDK.

Tópicos

Tópicos

- Criação de fila no Amazon SQS
- Atualizando filas do Amazon SQS
- Excluindo filas do Amazon SQS
- Enviando mensagens SMS do Amazon
- Recebimento de mensagens do Amazon SQS

Criação de fila no Amazon SQS

Este exemplo mostra como usar o AWS SDK para .NET para criar uma fila do Amazon SQS. O aplicação cria uma fila de mensagens não entregues se você não fornecer o ARN para uma. Em seguida, ele cria uma fila de mensagens padrão, que inclui uma fila de mensagens não entregues (a que você forneceu ou a que foi criada).

Se você não fornecer nenhum argumento de linha de comando, a aplicação simplesmente mostrará informações sobre todas as filas existentes.

As seções a seguir fornecem snippets desse exemplo. O código completo do exemplo é mostrado depois e pode ser criado e executado como está.

Tópicos

- Mostrar filas existentes
- Criação da fila
- Obtenha o ARN de uma fila
- Código completo

- Considerações adicionais

Mostrar filas existentes

O trecho a seguir mostra uma lista das filas existentes na região do cliente SQS e os atributos de cada fila.

O exemplo no final deste tópico mostra o snippet em uso.

```
//
// Method to show a list of the existing queues
private static async Task ShowQueues(IAmazonSQS sqsClient)
{
  ListQueuesResponse responseList = await sqsClient.ListQueuesAsync("");
  Console.WriteLine();
  foreach(string qUrl in responseList.QueueUrls)
  {
    // Get and show all attributes. Could also get a subset.
    await ShowAllAttributes(sqsClient, qUrl);
  }
}

//
// Method to show all attributes of a queue
private static async Task ShowAllAttributes(IAmazonSQS sqsClient, string qUrl)
{
  var attributes = new List<string>{ QueueAttributeName.All };
  GetQueueAttributesResponse responseGetAtt =
    await sqsClient.GetQueueAttributesAsync(qUrl, attributes);
  Console.WriteLine($"Queue: {qUrl}");
  foreach(var att in responseGetAtt.Attributes)
    Console.WriteLine($"\t{att.Key}: {att.Value}");
}
```

Criação da fila

O seguinte snippet cria uma fila. O trecho inclui o uso de uma fila de mensagens não entregues, mas uma fila de mensagens não entregues não é necessariamente necessária para suas filas.

O exemplo no final deste tópico mostra o snippet em uso.

```
//
// Method to create a queue. Returns the queue URL.
```

```
    private static async Task<string> CreateQueue(
      IAmazonSQS sqsClient, string qName, string deadLetterQueueUrl=null,
      string maxReceiveCount=null, string receiveWaitTime=null)
    {
      var attrs = new Dictionary<string, string>();

      // If a dead-letter queue is given, create a message queue
      if(!string.IsNullOrEmpty(deadLetterQueueUrl))
      {
        attrs.Add(QueueAttributeName.ReceiveMessageWaitTimeSeconds, receiveWaitTime);
        attrs.Add(QueueAttributeName.RedrivePolicy,
          $"{{\"deadLetterTargetArn\":\"{await GetQueueArn(sqsClient,
deadLetterQueueUrl)}\"," +
          $"\"maxReceiveCount\":\"{maxReceiveCount}\"}}");
        // Add other attributes for the message queue such as VisibilityTimeout
      }

      // If no dead-letter queue is given, create one of those instead
      //else
      //{
      //  // Add attributes for the dead-letter queue as needed
      //  attrs.Add();
      //}

      // Create the queue
      CreateQueueResponse responseCreate = await sqsClient.CreateQueueAsync(
          new CreateQueueRequest{QueueName = qName, Attributes = attrs});
      return responseCreate.QueueUrl;
    }
```

Obtenha o ARN de uma fila

O snippet a seguir obtém o ARN da fila identificada pelo URL da fila em questão.

O exemplo no final deste tópico mostra o snippet em uso.

```
    //
    // Method to get the ARN of a queue
    private static async Task<string> GetQueueArn(IAmazonSQS sqsClient, string qUrl)
    {
      GetQueueAttributesResponse responseGetAtt = await
sqsClient.GetQueueAttributesAsync(
        qUrl, new List<string>{QueueAttributeName.QueueArn});
      return responseGetAtt.QueueARN;
```

```
    }
```

Código completo

Esta seção mostra as referências relevantes e o código completo desse exemplo.

Referências do SDK

NuGet pacotes:

- AWSSDK.SQS

Elementos de programação:

- Namespace Amazon.SQS
  - Classe Amazon SQSClient
  - Classe QueueAttributeName
- Namespace Amazon.SQS.Model
  - Classe CreateQueueRequest
  - Classe CreateQueueResponse
  - Classe GetQueueAttributesResponse
  - Classe ListQueuesResponse

O código

```
using System;
using System.Threading.Tasks;
using System.Collections.Generic;
using Amazon.SQS;
using Amazon.SQS.Model;

namespace SQSCreateQueue
{
  // = = =
```

```
  class Program
  {
    private const string MaxReceiveCount = "10";
    private const string ReceiveMessageWaitTime = "2";
    private const int MaxArgs = 3;

    static async Task Main(string[] args)
    {
      // Parse the command line and show help if necessary
      var parsedArgs = CommandLine.Parse(args);
      if(parsedArgs.Count > MaxArgs)
        CommandLine.ErrorExit(
          "\nToo many command-line arguments.\nRun the command with no arguments to see
 help.");

      // Create the Amazon SQS client
      var sqsClient = new AmazonSQSClient();

      // In the case of no command-line arguments, just show help and the existing
 queues
      if(parsedArgs.Count == 0)
      {
        PrintHelp();
        Console.WriteLine("\nNo arguments specified.");
        Console.Write("Do you want to see a list of the existing queues? ((y) or n):
 ");
        string response = Console.ReadLine();
        if((string.IsNullOrEmpty(response)) || (response.ToLower() == "y"))
          await ShowQueues(sqsClient);
        return;
      }

      // Get the application arguments from the parsed list
      string queueName =
        CommandLine.GetArgument(parsedArgs, null, "-q", "--queue-name");
      string deadLetterQueueUrl =
        CommandLine.GetArgument(parsedArgs, null, "-d", "--dead-letter-queue");
      string maxReceiveCount =
        CommandLine.GetArgument(parsedArgs, MaxReceiveCount, "-m", "--max-receive-
count");
      string receiveWaitTime =
        CommandLine.GetArgument(parsedArgs, ReceiveMessageWaitTime, "-w", "--wait-
time");
```

```
      if(string.IsNullOrEmpty(queueName))
        CommandLine.ErrorExit(
          "\nYou must supply a queue name.\nRun the command with no arguments to see
help.");

      // If a dead-letter queue wasn't given, create one
      if(string.IsNullOrEmpty(deadLetterQueueUrl))
      {
        Console.WriteLine("\nNo dead-letter queue was specified. Creating one...");
        deadLetterQueueUrl = await CreateQueue(sqsClient, queueName + "__dlq");
        Console.WriteLine($"Your new dead-letter queue:");
        await ShowAllAttributes(sqsClient, deadLetterQueueUrl);
      }

      // Create the message queue
      string messageQueueUrl = await CreateQueue(
        sqsClient, queueName, deadLetterQueueUrl, maxReceiveCount, receiveWaitTime);
      Console.WriteLine($"Your new message queue:");
      await ShowAllAttributes(sqsClient, messageQueueUrl);
    }


    //
    // Method to show a list of the existing queues
    private static async Task ShowQueues(IAmazonSQS sqsClient)
    {
      ListQueuesResponse responseList = await sqsClient.ListQueuesAsync("");
      Console.WriteLine();
      foreach(string qUrl in responseList.QueueUrls)
      {
        // Get and show all attributes. Could also get a subset.
        await ShowAllAttributes(sqsClient, qUrl);
      }
    }


    //
    // Method to create a queue. Returns the queue URL.
    private static async Task<string> CreateQueue(
      IAmazonSQS sqsClient, string qName, string deadLetterQueueUrl=null,
      string maxReceiveCount=null, string receiveWaitTime=null)
    {
      var attrs = new Dictionary<string, string>();
```

```
      // If a dead-letter queue is given, create a message queue
      if(!string.IsNullOrEmpty(deadLetterQueueUrl))
      {
        attrs.Add(QueueAttributeName.ReceiveMessageWaitTimeSeconds, receiveWaitTime);
        attrs.Add(QueueAttributeName.RedrivePolicy,
          $"{{\"deadLetterTargetArn\":\"{await GetQueueArn(sqsClient,
deadLetterQueueUrl)}\"," +
          $"\"maxReceiveCount\":\"{maxReceiveCount}\"}}");
        // Add other attributes for the message queue such as VisibilityTimeout
      }

      // If no dead-letter queue is given, create one of those instead
      //else
      //{
      //  // Add attributes for the dead-letter queue as needed
      //  attrs.Add();
      //}

      // Create the queue
      CreateQueueResponse responseCreate = await sqsClient.CreateQueueAsync(
          new CreateQueueRequest{QueueName = qName, Attributes = attrs});
      return responseCreate.QueueUrl;
    }


    //
    // Method to get the ARN of a queue
    private static async Task<string> GetQueueArn(IAmazonSQS sqsClient, string qUrl)
    {
      GetQueueAttributesResponse responseGetAtt = await
sqsClient.GetQueueAttributesAsync(
        qUrl, new List<string>{QueueAttributeName.QueueArn});
      return responseGetAtt.QueueARN;
    }


    //
    // Method to show all attributes of a queue
    private static async Task ShowAllAttributes(IAmazonSQS sqsClient, string qUrl)
    {
      var attributes = new List<string>{ QueueAttributeName.All };
      GetQueueAttributesResponse responseGetAtt =
        await sqsClient.GetQueueAttributesAsync(qUrl, attributes);
      Console.WriteLine($"Queue: {qUrl}");
```

```
      foreach(var att in responseGetAtt.Attributes)
        Console.WriteLine($"\t{att.Key}: {att.Value}");
    }



    //
    // Command-line help
    private static void PrintHelp()
    {
      Console.WriteLine(
      "\nUsage: SQSCreateQueue -q <queue-name> [-d <dead-letter-queue>]" +
        " [-m <max-receive-count>] [-w <wait-time>]" +
      "\n  -q, --queue-name: The name of the queue you want to create." +
      "\n  -d, --dead-letter-queue: The URL of an existing queue to be used as the
dead-letter queue."+
      "\n      If this argument isn't supplied, a new dead-letter queue will be
created." +
      "\n  -m, --max-receive-count: The value for maxReceiveCount in the RedrivePolicy
of the queue." +
      $"\n      Default is {MaxReceiveCount}." +
      "\n  -w, --wait-time: The value for ReceiveMessageWaitTimeSeconds of the queue
for long polling." +
      $"\n      Default is {ReceiveMessageWaitTime}.");
    }
  }



  // = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = =
  // Class that represents a command line on the console or terminal.
  // (This is the same for all examples. When you have seen it once, you can ignore
it.)
  static class CommandLine
  {
    //
    // Method to parse a command line of the form: "--key value" or "-k value".
    //
    // Parameters:
    // - args: The command-line arguments passed into the application by the system.
    //
    // Returns:
    // A Dictionary with string Keys and Values.
    //
    // If a key is found without a matching value, Dictionary.Value is set to the key
```

```
    //  (including the dashes).
    // If a value is found without a matching key, Dictionary.Key is set to "--NoKeyN",
    //  where "N" represents sequential numbers.
    public static Dictionary<string,string> Parse(string[] args)
    {
      var parsedArgs = new Dictionary<string,string>();
      int i = 0, n = 0;
      while(i < args.Length)
      {
        // If the first argument in this iteration starts with a dash it's an option.
        if(args[i].StartsWith("-"))
        {
          var key = args[i++];
          var value = key;

          // Check to see if there's a value that goes with this option?
          if((i < args.Length) && (!args[i].StartsWith("-"))) value = args[i++];
          parsedArgs.Add(key, value);
        }

        // If the first argument in this iteration doesn't start with a dash, it's a
value
        else
        {
          parsedArgs.Add("--NoKey" + n.ToString(), args[i++]);
          n++;
        }
      }

      return parsedArgs;
    }

    //
    // Method to get an argument from the parsed command-line arguments
    //
    // Parameters:
    // - parsedArgs: The Dictionary object returned from the Parse() method (shown
above).
    // - defaultValue: The default string to return if the specified key isn't in
parsedArgs.
    // - keys: An array of keys to look for in parsedArgs.
    public static string GetArgument(
      Dictionary<string,string> parsedArgs, string defaultReturn, params string[] keys)
    {
```

```
      string retval = null;
      foreach(var key in keys)
        if(parsedArgs.TryGetValue(key, out retval)) break;
      return retval ?? defaultReturn;
    }

    //
    // Method to exit the application with an error.
    public static void ErrorExit(string msg, int code=1)
    {
      Console.WriteLine("\nError");
      Console.WriteLine(msg);
      Environment.Exit(code);
    }
  }

}
```

Considerações adicionais

- O nome da fila deve ser composto de caracteres alfanuméricos, hifens e sublinhados.
- Os nomes das filas e as filas diferenciam maiúsculas de URLs minúsculas
- Se precisar do URL da fila, mas tiver apenas o nome da fila, use um dos `AmazonSQSClient.GetQueueUrlAsync` métodos.
- Para obter informações sobre os vários atributos de fila que você pode definir, consulte CreateQueueRequestna Referência da AWS SDK para .NET API ou SetQueueAttributesna Referência da API do Amazon Simple Queue Service.
- Este exemplo especifica uma sondagem longa para todas as mensagens na fila que você cria. Isto é feito usando o atributo `ReceiveMessageWaitTimeSeconds`.

  Você também pode especificar uma sondagem longa durante uma chamada para os `ReceiveMessageAsync` métodos da SQSClient classe Amazon. Para obter mais informações, consulte Recebimento de mensagens do Amazon SQS.

Para obter informações sobre sondagem curta versus sondagem longa, consulte Sondagem curta e longa no Guia do desenvolvedor do Amazon Simple Queue Service.

- Uma fila de mensagens não entregues é aquela fila que outras filas (de origem) podem direcionar para mensagens que não foram processadas com sucesso. Para obter mais informações, consulte filas de mensagens não entregues do Amazon SQS no Guia do desenvolvedor do Amazon Simple Queue Service.

- Você também pode ver a lista de filas e os resultados desse exemplo no console do Amazon SQS.

Atualizando filas do Amazon SQS

Este exemplo mostra como usar o AWS SDK para .NET para atualizar uma fila do Amazon SQS. Depois de algumas verificações, a aplicação atualiza o atributo fornecido com o valor fornecido e, em seguida, mostra todos os atributos da fila.

Se somente o URL da fila estiver incluído nos argumentos da linha de comando, a aplicação simplesmente mostrará todos os atributos da fila.

As seções a seguir fornecem snippets desse exemplo. O código completo do exemplo é mostrado depois e pode ser criado e executado como está.

Tópicos

- Mostrar atributos da fila
- Validar nome do atributo
- Atualizar atributo de fila
- Código completo
- Considerações adicionais

Mostrar atributos da fila

O snippet a seguir mostra os atributos da fila identificados pelo URL da fila em questão.

O exemplo no final deste tópico mostra o snippet em uso.

```
    //
```

```
    // Method to show all attributes of a queue
    private static async Task ShowAllAttributes(IAmazonSQS sqsClient, string qUrl)
    {
      GetQueueAttributesResponse responseGetAtt =
        await sqsClient.GetQueueAttributesAsync(qUrl,
          new List<string>{ QueueAttributeName.All });
      Console.WriteLine($"Queue: {qUrl}");
      foreach(var att in responseGetAtt.Attributes)
        Console.WriteLine($"\t{att.Key}: {att.Value}");
    }
```

Validar nome do atributo

O snippet a seguir valida o nome do atributo que está sendo atualizado.

O exemplo no final deste tópico mostra o snippet em uso.

```
    //
    // Method to check the name of the attribute
    private static bool ValidAttribute(string attribute)
    {
      var attOk = false;
      var qAttNameType = typeof(QueueAttributeName);
      List<string> qAttNamefields = new List<string>();
      foreach(var field in qAttNameType.GetFields())
       qAttNamefields.Add(field.Name);
      foreach(var name in qAttNamefields)
        if(attribute == name) { attOk = true; break; }
      return attOk;
    }
```

Atualizar atributo de fila

O snippet a seguir atualiza um atributo da fila identificado pelo URL da fila em questão.

O exemplo no final deste tópico mostra o snippet em uso.

```
    //
    // Method to update a queue attribute
    private static async Task UpdateAttribute(
      IAmazonSQS sqsClient, string qUrl, string attribute, string value)
    {
      await sqsClient.SetQueueAttributesAsync(qUrl,
        new Dictionary<string, string>{{attribute, value}});
```

```
    }
```

Código completo

Esta seção mostra as referências relevantes e o código completo desse exemplo.

Referências do SDK

NuGet pacotes:

- AWSSDK.SQS

Elementos de programação:

- Namespace Amazon.SQS

  Classe Amazon SQSClient

  Classe QueueAttributeName
- Namespace Amazon.SQS.Model

  Classe GetQueueAttributesResponse

O código

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.SQS;
using Amazon.SQS.Model;

namespace SQSUpdateQueue
{
  // = = =
```

```
    {
      // Parse the command line and show help if necessary
      var parsedArgs = CommandLine.Parse(args);
      if(parsedArgs.Count == 0)
      {
        PrintHelp();
        return;
      }
      if((parsedArgs.Count > MaxArgs) || (parsedArgs.Count == InvalidArgCount))
        CommandLine.ErrorExit("\nThe number of command-line arguments is incorrect." +
          "\nRun the command with no arguments to see help.");

      // Get the application arguments from the parsed list
      var qUrl = CommandLine.GetArgument(parsedArgs, null, "-q");
      var attribute = CommandLine.GetArgument(parsedArgs, null, "-a");
      var value = CommandLine.GetArgument(parsedArgs, null, "-v", "--value");

      if(string.IsNullOrEmpty(qUrl))
        CommandLine.ErrorExit("\nYou must supply at least a queue URL." +
          "\nRun the command with no arguments to see help.");

      // Create the Amazon SQS client
      var sqsClient = new AmazonSQSClient();

      // In the case of one command-line argument, just show the attributes for the
queue
      if(parsedArgs.Count == 1)
        await ShowAllAttributes(sqsClient, qUrl);

      // Otherwise, attempt to update the given queue attribute with the given value
      else
      {
        // Check to see if the attribute is valid
        if(ValidAttribute(attribute))
        {
          // Perform the update and then show all the attributes of the queue
          await UpdateAttribute(sqsClient, qUrl, attribute, value);
          await ShowAllAttributes(sqsClient, qUrl);
        }
        else
        {
          Console.WriteLine($"\nThe given attribute name, {attribute}, isn't valid.");
        }
      }
```

```
    }


    //
    // Method to show all attributes of a queue
    private static async Task ShowAllAttributes(IAmazonSQS sqsClient, string qUrl)
    {
      GetQueueAttributesResponse responseGetAtt =
        await sqsClient.GetQueueAttributesAsync(qUrl,
          new List<string>{ QueueAttributeName.All });
      Console.WriteLine($"Queue: {qUrl}");
      foreach(var att in responseGetAtt.Attributes)
        Console.WriteLine($"\t{att.Key}: {att.Value}");
    }


    //
    // Method to check the name of the attribute
    private static bool ValidAttribute(string attribute)
    {
      var attOk = false;
      var qAttNameType = typeof(QueueAttributeName);
      List<string> qAttNamefields = new List<string>();
      foreach(var field in qAttNameType.GetFields())
       qAttNamefields.Add(field.Name);
      foreach(var name in qAttNamefields)
        if(attribute == name) { attOk = true; break; }
      return attOk;
    }


    //
    // Method to update a queue attribute
    private static async Task UpdateAttribute(
      IAmazonSQS sqsClient, string qUrl, string attribute, string value)
    {
      await sqsClient.SetQueueAttributesAsync(qUrl,
        new Dictionary<string, string>{{attribute, value}});
    }


    //
    // Command-line help
    private static void PrintHelp()
```

```
    {
      Console.WriteLine("\nUsage: SQSUpdateQueue -q queue_url [-a attribute -v
value]");
      Console.WriteLine("  -q: The URL of the queue you want to update.");
      Console.WriteLine("  -a: The name of the attribute to update.");
      Console.WriteLine("  -v, --value: The value to assign to the attribute.");
    }
  }


  // = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
= = =
  // Class that represents a command line on the console or terminal.
  // (This is the same for all examples. When you have seen it once, you can ignore
it.)
  static class CommandLine
  {
    //
    // Method to parse a command line of the form: "--key value" or "-k value".
    //
    // Parameters:
    // - args: The command-line arguments passed into the application by the system.
    //
    // Returns:
    // A Dictionary with string Keys and Values.
    //
    // If a key is found without a matching value, Dictionary.Value is set to the key
    //  (including the dashes).
    // If a value is found without a matching key, Dictionary.Key is set to "--NoKeyN",
    //  where "N" represents sequential numbers.
    public static Dictionary<string,string> Parse(string[] args)
    {
      var parsedArgs = new Dictionary<string,string>();
      int i = 0, n = 0;
      while(i < args.Length)
      {
        // If the first argument in this iteration starts with a dash it's an option.
        if(args[i].StartsWith("-"))
        {
          var key = args[i++];
          var value = key;

          // Check to see if there's a value that goes with this option?
          if((i < args.Length) && (!args[i].StartsWith("-"))) value = args[i++];
```

```
          parsedArgs.Add(key, value);
        }

        // If the first argument in this iteration doesn't start with a dash, it's a
value
        else
        {
          parsedArgs.Add("--NoKey" + n.ToString(), args[i++]);
          n++;
        }
      }

      return parsedArgs;
    }

    //
    // Method to get an argument from the parsed command-line arguments
    //
    // Parameters:
    // - parsedArgs: The Dictionary object returned from the Parse() method (shown
above).
    // - defaultValue: The default string to return if the specified key isn't in
parsedArgs.
    // - keys: An array of keys to look for in parsedArgs.
    public static string GetArgument(
      Dictionary<string,string> parsedArgs, string defaultReturn, params string[] keys)
    {
      string retval = null;
      foreach(var key in keys)
        if(parsedArgs.TryGetValue(key, out retval)) break;
      return retval ?? defaultReturn;
    }

    //
    // Method to exit the application with an error.
    public static void ErrorExit(string msg, int code=1)
    {
      Console.WriteLine("\nError");
      Console.WriteLine(msg);
      Environment.Exit(code);
    }
  }
```

```
}
```

Considerações adicionais

- Para atualizar o `RedrivePolicy` atributo, você deve cotar o valor inteiro e escapar das aspas dos key/value pares, conforme apropriado para seu sistema operacional.

  No Windows, por exemplo, o valor é construído de forma semelhante ao seguinte:

  ```
  "{\"deadLetterTargetArn\":\"DEAD_LETTER-QUEUE-ARN\",\"maxReceiveCount\":\"10\"}"
  ```

Excluindo filas do Amazon SQS

Este exemplo mostra como usar o AWS SDK para .NET para excluir uma fila do Amazon SQS. A explicação exclui a fila, espera até um determinado período de tempo até que a fila seja excluída e, em seguida, mostra uma lista de filas restantes.

Se você não fornecer nenhum argumento de linha de comando, a aplicação simplesmente mostrará uma lista de filas existentes.

As seções a seguir fornecem snippets desse exemplo. O código completo do exemplo é mostrado depois e pode ser criado e executado como está.

Tópicos

- Excluir a fila
- Aguardar até a fila ser removida
- Mostrar uma lista de filas existentes
- Código completo
- Considerações adicionais

Excluir a fila

O snippet a seguir exclui a fila identificada pelo URL da fila em questão.

O exemplo no final deste tópico mostra o snippet em uso.

```
    //
```

```
    // Method to delete an SQS queue
    private static async Task DeleteQueue(IAmazonSQS sqsClient, string qUrl)
    {
      Console.WriteLine($"Deleting queue {qUrl}...");
      await sqsClient.DeleteQueueAsync(qUrl);
      Console.WriteLine($"Queue {qUrl} has been deleted.");
    }
```

Aguardar até a fila ser removida

O snippet a seguir aguarda a conclusão do processo de exclusão, o que pode levar até 60 segundos.

O exemplo no final deste tópico mostra o snippet em uso.

```
    //
    // Method to wait up to a given number of seconds
    private static async Task Wait(
      IAmazonSQS sqsClient, int numSeconds, string qUrl)
    {
      Console.WriteLine($"Waiting for up to {numSeconds} seconds.");
      Console.WriteLine("Press any key to stop waiting. (Response might be slightly
delayed.)");
      for(int i=0; i<numSeconds; i++)
      {
        Console.Write(".");
        Thread.Sleep(1000);
        if(Console.KeyAvailable) break;

        // Check to see if the queue is gone yet
        var found = false;
        ListQueuesResponse responseList = await sqsClient.ListQueuesAsync("");
        foreach(var url in responseList.QueueUrls)
        {
          if(url == qUrl)
          {
            found = true;
            break;
          }
        }
        if(!found) break;
      }
    }
```

Mostrar uma lista de filas existentes

O trecho a seguir mostra uma lista das filas existentes na região do cliente SQS.

O exemplo no final deste tópico mostra o snippet em uso.

```
    //
    // Method to show a list of the existing queues
    private static async Task ListQueues(IAmazonSQS sqsClient)
    {
      ListQueuesResponse responseList = await sqsClient.ListQueuesAsync("");
      Console.WriteLine("\nList of queues:");
      foreach(var qUrl in responseList.QueueUrls)
        Console.WriteLine($"- {qUrl}");
    }
```

Código completo

Esta seção mostra as referências relevantes e o código completo desse exemplo.

Referências do SDK

NuGet pacotes:

- AWSSDK.SQS

Elementos de programação:

- Namespace Amazon.SQS

  Classe Amazon SQSClient
- Namespace Amazon.SQS.Model

  Classe ListQueuesResponse

O código

```
using System;
using System.Threading;
using System.Threading.Tasks;
using Amazon.SQS;
using Amazon.SQS.Model;
```

```
namespace SQSDeleteQueue
{
  // = = =
```

```
    }


    //
    // Method to delete an SQS queue
    private static async Task DeleteQueue(IAmazonSQS sqsClient, string qUrl)
    {
      Console.WriteLine($"Deleting queue {qUrl}...");
      await sqsClient.DeleteQueueAsync(qUrl);
      Console.WriteLine($"Queue {qUrl} has been deleted.");
    }


    //
    // Method to wait up to a given number of seconds
    private static async Task Wait(
      IAmazonSQS sqsClient, int numSeconds, string qUrl)
    {
      Console.WriteLine($"Waiting for up to {numSeconds} seconds.");
      Console.WriteLine("Press any key to stop waiting. (Response might be slightly
delayed.)");
      for(int i=0; i<numSeconds; i++)
      {
        Console.Write(".");
        Thread.Sleep(1000);
        if(Console.KeyAvailable) break;

        // Check to see if the queue is gone yet
        var found = false;
        ListQueuesResponse responseList = await sqsClient.ListQueuesAsync("");
        foreach(var url in responseList.QueueUrls)
        {
          if(url == qUrl)
          {
            found = true;
            break;
          }
        }
        if(!found) break;
      }
    }


    //
```

```
    // Method to show a list of the existing queues
    private static async Task ListQueues(IAmazonSQS sqsClient)
    {
      ListQueuesResponse responseList = await sqsClient.ListQueuesAsync("");
      Console.WriteLine("\nList of queues:");
      foreach(var qUrl in responseList.QueueUrls)
        Console.WriteLine($"- {qUrl}");
    }
  }
}
```

Considerações adicionais

- A chamada de API `DeleteQueueAsync` não verifica se a fila que você está excluindo está sendo usada como uma fila de mensagens não entregues. Um procedimento mais sofisticado poderia verificar isso.

- Você também pode ver a lista de filas e os resultados desse exemplo no console do Amazon SQS.

Enviando mensagens SMS do Amazon

Este exemplo mostra como usar o para enviar mensagens AWS SDK para .NET para uma fila do Amazon SQS, que você pode criar programaticamente ou usando o console do Amazon SQS. A aplicação envia uma única mensagem para a fila e, em seguida, um lote de mensagens. Em seguida, a aplicação aguarda a entrada do usuário, que pode ser mensagens adicionais para enviar para a fila ou uma solicitação para sair da aplicação.

Este exemplo e o próximo exemplo sobre o recebimento de mensagens podem ser usados juntos para ver o fluxo de mensagens no Amazon SQS.

As seções a seguir fornecem snippets desse exemplo. O código completo do exemplo é mostrado depois e pode ser criado e executado como está.

Tópicos

- Enviar uma mensagem
- Enviar um lote de mensagens
- Excluir todas as mensagens da fila
- Código completo

- Considerações adicionais

Enviar uma mensagem

O snippet a seguir envia uma mensagem para a fila identificada pelo URL da fila em questão.

O exemplo no final deste tópico mostra o snippet em uso.

```
//
// Method to put a message on a queue
// Could be expanded to include message attributes, etc., in a SendMessageRequest
private static async Task SendMessage(
  IAmazonSQS sqsClient, string qUrl, string messageBody)
{
  SendMessageResponse responseSendMsg =
    await sqsClient.SendMessageAsync(qUrl, messageBody);
  Console.WriteLine($"Message added to queue\n  {qUrl}");
  Console.WriteLine($"HttpStatusCode: {responseSendMsg.HttpStatusCode}");
}
```

Enviar um lote de mensagens

O snippet a seguir envia um lote de mensagens para a fila identificada pelo URL da fila em questão.

O exemplo no final deste tópico mostra o snippet em uso.

```
//
// Method to put a batch of messages on a queue
// Could be expanded to include message attributes, etc.,
// in the SendMessageBatchRequestEntry objects
private static async Task SendMessageBatch(
  IAmazonSQS sqsClient, string qUrl, List<SendMessageBatchRequestEntry> messages)
{
  Console.WriteLine($"\nSending a batch of messages to queue\n  {qUrl}");
  SendMessageBatchResponse responseSendBatch =
    await sqsClient.SendMessageBatchAsync(qUrl, messages);
  // Could test responseSendBatch.Failed here
  foreach(SendMessageBatchResultEntry entry in responseSendBatch.Successful)
    Console.WriteLine($"Message {entry.Id} successfully queued.");
}
```

Excluir todas as mensagens da fila

O snippet a seguir exclui todas as mensagens da fila identificada pelo URL da fila em questão. Isso também é conhecido como limpando a fila.

O exemplo no final deste tópico mostra o snippet em uso.

```
//
// Method to delete all messages from the queue
private static async Task DeleteAllMessages(IAmazonSQS sqsClient, string qUrl)
{
  Console.WriteLine($"\nPurging messages from queue\n  {qUrl}...");
  PurgeQueueResponse responsePurge = await sqsClient.PurgeQueueAsync(qUrl);
  Console.WriteLine($"HttpStatusCode: {responsePurge.HttpStatusCode}");
}
```

Código completo

Esta seção mostra as referências relevantes e o código completo desse exemplo.

Referências do SDK

NuGet pacotes:

- AWSSDK.SQS

Elementos de programação:

- Namespace Amazon.SQS

  Classe Amazon SQSClient
- Namespace Amazon.SQS.Model

  Classe PurgeQueueResponse

  Classe SendMessageBatchResponse

  Classe SendMessageResponse

  Classe SendMessageBatchRequestEntry

  Classe SendMessageBatchResultEntry

O código

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.SQS;
using Amazon.SQS.Model;

namespace SQSSendMessages
{
  // = = =
```

```
    // Send some example messages to the given queue
    // A single message
    await SendMessage(sqsClient, args[0], JsonMessage);

    // A batch of messages
    var batchMessages = new List<SendMessageBatchRequestEntry>{
      new SendMessageBatchRequestEntry("xmlMsg", XmlMessage),
      new SendMessageBatchRequestEntry("customeMsg", CustomMessage),
      new SendMessageBatchRequestEntry("textMsg", TextMessage)};
    await SendMessageBatch(sqsClient, args[0], batchMessages);

    // Let the user send their own messages or quit
    await InteractWithUser(sqsClient, args[0]);

    // Delete all messages that are still in the queue
    await DeleteAllMessages(sqsClient, args[0]);
  }


  //
  // Method to put a message on a queue
  // Could be expanded to include message attributes, etc., in a SendMessageRequest
  private static async Task SendMessage(
    IAmazonSQS sqsClient, string qUrl, string messageBody)
  {
    SendMessageResponse responseSendMsg =
      await sqsClient.SendMessageAsync(qUrl, messageBody);
    Console.WriteLine($"Message added to queue\n  {qUrl}");
    Console.WriteLine($"HttpStatusCode: {responseSendMsg.HttpStatusCode}");
  }


  //
  // Method to put a batch of messages on a queue
  // Could be expanded to include message attributes, etc.,
  // in the SendMessageBatchRequestEntry objects
  private static async Task SendMessageBatch(
    IAmazonSQS sqsClient, string qUrl, List<SendMessageBatchRequestEntry> messages)
  {
    Console.WriteLine($"\nSending a batch of messages to queue\n  {qUrl}");
    SendMessageBatchResponse responseSendBatch =
      await sqsClient.SendMessageBatchAsync(qUrl, messages);
    // Could test responseSendBatch.Failed here
    foreach(SendMessageBatchResultEntry entry in responseSendBatch.Successful)
```

```
        Console.WriteLine($"Message {entry.Id} successfully queued.");
    }


    //
    // Method to get input from the user
    // They can provide messages to put in the queue or exit the application
    private static async Task InteractWithUser(IAmazonSQS sqsClient, string qUrl)
    {
      string response;
      while (true)
      {
        // Get the user's input
        Console.WriteLine("\nType a message for the queue or \"exit\" to quit:");
        response = Console.ReadLine();
        if(response.ToLower() == "exit") break;

        // Put the user's message in the queue
        await SendMessage(sqsClient, qUrl, response);
      }
    }


    //
    // Method to delete all messages from the queue
    private static async Task DeleteAllMessages(IAmazonSQS sqsClient, string qUrl)
    {
      Console.WriteLine($"\nPurging messages from queue\n  {qUrl}...");
      PurgeQueueResponse responsePurge = await sqsClient.PurgeQueueAsync(qUrl);
      Console.WriteLine($"HttpStatusCode: {responsePurge.HttpStatusCode}");
    }
  }
}
```

Considerações adicionais

- Para obter informações sobre várias limitações nas mensagens, incluindo os caracteres permitidos, consulte Cotas relacionadas às mensagens no Guia do desenvolvedor do Amazon Simple Queue Service.

- As mensagens permanecem nas filas até serem excluídas ou a fila ser removida. Quando uma mensagem for recebida por uma aplicação, ela não ficará visível na fila, mesmo que ainda exista

na fila. Para obter mais informações sobre tempos limite de visibilidade, consulte Tempo limite de visibilidade do Amazon SQS.

- Além do corpo da mensagem, você também pode adicionar atributos às mensagens. Para obter mais informações, consulte Mensagem de metadados.

Recebimento de mensagens do Amazon SQS

Este exemplo mostra como usar o AWS SDK para .NET para receber mensagens de uma fila do Amazon SQS, que você pode criar programaticamente ou usando o console do Amazon SQS. A aplicação lê uma única mensagem da fila, processa a mensagem (nesse caso, exibe o corpo da mensagem no console) e, em seguida, exclui a mensagem da fila. A aplicação repete essas etapas até que o usuário digite uma tecla no teclado.

Este exemplo e o exemplo anterior sobre envio de mensagens, podem ser usados juntos para ver o fluxo de mensagens no Amazon SQS.

As seções a seguir fornecem snippets desse exemplo. O código completo do exemplo é mostrado depois e pode ser criado e executado como está.

Tópicos

- Recebimento de uma mensagem
- Excluir uma mensagem
- Código completo
- Considerações adicionais

Recebimento de uma mensagem

O trecho a seguir recebe uma mensagem da fila identificada pelo URL da fila em questão.

O exemplo no final deste tópico mostra o snippet em uso.

```
//
// Method to read a message from the given queue
// In this example, it gets one message at a time
private static async Task<ReceiveMessageResponse> GetMessage(
  IAmazonSQS sqsClient, string qUrl, int waitTime=0)
{
```

```
    return await sqsClient.ReceiveMessageAsync(new ReceiveMessageRequest{
      QueueUrl=qUrl,
      MaxNumberOfMessages=MaxMessages,
      WaitTimeSeconds=waitTime
      // (Could also request attributes, set visibility timeout, etc.)
    });
  }
```

Excluir uma mensagem

O snippet a seguir exclui uma mensagem da fila identificada pelo URL da fila em questão.

O exemplo no final deste tópico mostra o snippet em uso.

```
  //
  // Method to delete a message from a queue
  private static async Task DeleteMessage(
    IAmazonSQS sqsClient, Message message, string qUrl)
  {
    Console.WriteLine($"\nDeleting message {message.MessageId} from queue...");
    await sqsClient.DeleteMessageAsync(qUrl, message.ReceiptHandle);
  }
```

Código completo

Esta seção mostra as referências relevantes e o código completo desse exemplo.

Referências do SDK

NuGet pacotes:

- AWSSDK.SQS

Elementos de programação:

- Namespace Amazon.SQS

  Classe Amazon SQSClient
- Namespace Amazon.SQS.Model

  Classe ReceiveMessageRequest

  Classe ReceiveMessageResponse

O código

```
using System;
using System.Threading.Tasks;
using Amazon.SQS;
using Amazon.SQS.Model;

namespace SQSReceiveMessages
{
  class Program
  {
    private const int MaxMessages = 1;
    private const int WaitTime = 2;
    static async Task Main(string[] args)
    {
      // Do some checks on the command-line
      if(args.Length == 0)
      {
        Console.WriteLine("\nUsage: SQSReceiveMessages queue_url");
        Console.WriteLine("   queue_url - The URL of an existing SQS queue.");
        return;
      }
      if(!args[0].StartsWith("https://sqs."))
      {
        Console.WriteLine("\nThe command-line argument isn't a queue URL:");
        Console.WriteLine($"{args[0]}");
        return;
      }

      // Create the Amazon SQS client
      var sqsClient = new AmazonSQSClient();

      // (could verify that the queue exists)
      // Read messages from the queue and perform appropriate actions
      Console.WriteLine($"Reading messages from queue\n  {args[0]}");
      Console.WriteLine("Press any key to stop. (Response might be slightly
 delayed.)");
      do
      {
        var msg = await GetMessage(sqsClient, args[0], WaitTime);
        if(msg.Messages.Count != 0)
        {
          if(ProcessMessage(msg.Messages[0]))
            await DeleteMessage(sqsClient, msg.Messages[0], args[0]);
```

```
        }
      } while(!Console.KeyAvailable);
    }


    //
    // Method to read a message from the given queue
    // In this example, it gets one message at a time
    private static async Task<ReceiveMessageResponse> GetMessage(
      IAmazonSQS sqsClient, string qUrl, int waitTime=0)
    {
      return await sqsClient.ReceiveMessageAsync(new ReceiveMessageRequest{
        QueueUrl=qUrl,
        MaxNumberOfMessages=MaxMessages,
        WaitTimeSeconds=waitTime
        // (Could also request attributes, set visibility timeout, etc.)
      });
    }


    //
    // Method to process a message
    // In this example, it simply prints the message
    private static bool ProcessMessage(Message message)
    {
      Console.WriteLine($"\nMessage body of {message.MessageId}:");
      Console.WriteLine($"{message.Body}");
      return true;
    }


    //
    // Method to delete a message from a queue
    private static async Task DeleteMessage(
      IAmazonSQS sqsClient, Message message, string qUrl)
    {
      Console.WriteLine($"\nDeleting message {message.MessageId} from queue...");
      await sqsClient.DeleteMessageAsync(qUrl, message.ReceiptHandle);
    }
  }
}
```

Considerações adicionais

- Para especificar uma sondagem longa, este exemplo usa a propriedade `WaitTimeSeconds` de cada chamada ao método `ReceiveMessageAsync`.

  Você também pode especificar uma sondagem longa para todas as mensagens em uma fila usando o atributo `ReceiveMessageWaitTimeSeconds` ao criar ou atualizar a fila.

  Para obter informações sobre sondagem curta versus sondagem longa, consulte Sondagem curta e longa no Guia do desenvolvedor do Amazon Simple Queue Service.

- Durante o processamento da mensagem, você pode usar o identificador de recebimento para alterar o tempo limite de visibilidade da mensagem. Para obter informações sobre como fazer isso, consulte os `ChangeMessageVisibilityAsync` métodos da SQSClient classe Amazon.

- Chamar o método `DeleteMessageAsync` remove a mensagem da fila incondicionalmente, qualquer que seja a configuração do tempo limite de visibilidade.

Usando AWS Lambda para serviço de computação

O AWS SDK para .NET suporte AWS Lambda, que permite executar código sem provisionar ou gerenciar servidores. Para obter mais informações, consulte a página do produto AWS Lambda e o Guia do desenvolvedor do AWS Lambda, especialmente a seção Trabalhando com C#.

APIs

Os AWS SDK para .NET suprimentos APIs para AWS Lambda. Eles APIs permitem que você trabalhe com recursos do Lambda, como funções, acionadores e eventos. Para ver o conjunto completo de APIs, consulte Lambda na Referência da AWS SDK para .NET API.

Os Lambda APIs são fornecidos por NuGet pacotes.

Pré-requisitos

Antes de começar, certifique-se de ter configurado seu ambiente e seu projeto. Revise também as informações em Uso da SDK.

Mais informações

Consulte Integração AWS com o.NET Aspire para obter informações sobre como desenvolver com AWS Lambda o.NET Aspire.

Tópicos

Tópicos

- Usando anotações para escrever funções AWS Lambda

Usando anotações para escrever funções AWS Lambda

Ao escrever funções do Lambda, às vezes é necessário escrever uma grande quantidade de código de manipulador e atualizar modelos do AWS CloudFormation , entre outras tarefas. Lambda Annotations é uma estrutura para ajudar a aliviar essas cargas para funções do Lambda do .NET 6, tornando assim a experiência de escrever Lambda em C# mais natural.

Como exemplo do benefício de usar a estrutura do Lambda Annotations, considere os seguintes trechos de código que adicionam dois números.

Sem Lambda Annotations

```
public class Functions
{
    public APIGatewayProxyResponse LambdaMathPlus(APIGatewayProxyRequest request,
 ILambdaContext context)
    {
        if (!request.PathParameters.TryGetValue("x", out var xs))
        {
            return new APIGatewayProxyResponse
            {
                StatusCode = (int)HttpStatusCode.BadRequest
            };
        }
        if (!request.PathParameters.TryGetValue("y", out var ys))
        {
            return new APIGatewayProxyResponse
            {
                StatusCode = (int)HttpStatusCode.BadRequest
            };
```

```
        }

        var x = int.Parse(xs);
        var y = int.Parse(ys);

        return new APIGatewayProxyResponse
        {
            StatusCode = (int)HttpStatusCode.OK,
            Body = (x + y).ToString(),
            Headers = new Dictionary<string, string> { { "Content-Type", "text/
plain" } }
        };
    }
}
```

Com Lambda Annotations

```
public class Functions
{
    [LambdaFunction]
    [RestApi("/plus/{x}/{y}")]
    public int Plus(int x, int y)
    {
        return x + y;
    }
}
```

Conforme mostrado no exemplo, o Lambda Annotations pode eliminar a necessidade de um determinado código clichê.

Para obter detalhes sobre como usar a estrutura, bem como informações adicionais, consulte os seguintes recursos:

- O GitHub README para documentação sobre os atributos APIs e as anotações do Lambda.
- A publicação no blog do Lambda Annotations.
- O pacote `Amazon.Lambda.Annotations` NuGet .
- O projeto Photo Asset Management começou GitHub. Especificamente, consulte a PamApiAnnotationspasta e as referências às anotações do Lambda no README do projeto.

> **Note**
>
> O exemplo anterior é específico para a V3 do. AWS SDK para .NET Se você usar o exemplo com a V4 do SDK (a versão mais recente), talvez seja necessário fazer ajustes de acordo com as informações em. Migrando para a versão 4

Bibliotecas e estruturas de alto nível para o AWS SDK para .NET

As seções a seguir contêm informações sobre bibliotecas e estruturas de alto nível que não fazem parte da funcionalidade principal do SDK. Essas bibliotecas e estruturas usam a funcionalidade principal do SDK para criar recursos que facilitam determinadas tarefas.

Se você é novo no AWS SDK para .NET, talvez queira conferir o Criando um aplicativo simples tópico primeiro. Ele fornece uma introdução ao SDK.

Antes de começar, certifique-se de ter configurado seu ambiente e seu projeto. Revise também as informações em Uso da SDK.

Tópicos

- AWS Estrutura de processamento de mensagens para.NET
- Integração AWS com o.NET Aspire no AWS SDK para .NET

AWS Estrutura de processamento de mensagens para.NET

A Estrutura de Processamento de AWS Mensagens para.NET é uma estrutura AWS nativa que simplifica o desenvolvimento de aplicativos de processamento de mensagens.NET que usam AWS serviços como Amazon Simple Queue Service (SQS), Amazon Simple Notification Service (SNS) e Amazon. EventBridge A estrutura reduz a quantidade de código padronizado que os desenvolvedores precisam escrever, permitindo que você se concentre na lógica empresarial ao publicar e consumir mensagens. Para obter detalhes sobre como a estrutura pode simplificar seu desenvolvimento, consulte a postagem do blog Apresentando a estrutura de processamento de AWS mensagens para.NET (versão prévia). A primeira parte, em particular, fornece uma demonstração que mostra a diferença entre usar chamadas de API de baixo nível e usar a estrutura.

O Message Processing Framework oferece suporte às seguintes atividades e recursos:

- Enviando mensagens para o SQS e publicando eventos no SNS e. EventBridge
- Receber e manipular mensagens do SQS usando um poller de longa duração, que normalmente é usado em serviços em segundo plano. Isso inclui gerenciar o tempo limite de visibilidade enquanto uma mensagem está sendo processada para evitar que outros clientes a processem.
- Manipulação de mensagens em AWS Lambda funções.
- Filas SQS FIFO (first-in-first-out) e tópicos de SNS.
- OpenTelemetry para registro.

Para obter detalhes sobre essas atividades e recursos, consulte a seção Recursos da postagem do blog e os tópicos listados abaixo.

Antes de começar, verifique se você configurou seu ambiente e configurou seu projeto. Revise também as informações em Uso da SDK.

Recursos adicionais

- O `AWS.Messaging`pacote em NuGet.org.
- A referência da API.
- O `README` arquivo no GitHub repositório em https://github.com/aws/aws-dotnet-messaging/
- Injeção de dependência.NET da Microsoft.
- Host genérico.NET da Microsoft.

Tópicos

- Comece a usar o AWS Message Processing Framework para.NET
- Publique mensagens com o AWS Message Processing Framework para.NET
- Consuma mensagens com o AWS Message Processing Framework para.NET
- Usando o FIFO com a estrutura de processamento de AWS mensagens para.NET
- Registro e telemetria aberta para a estrutura de processamento de AWS mensagens para.NET
- Personalize a estrutura de processamento de AWS mensagens para.NET
- Segurança para a estrutura de processamento de AWS mensagens para.NET

Comece a usar o AWS Message Processing Framework para.NET

Antes de começar, certifique-se de ter configurado seu ambiente e seu projeto. Revise também as informações em Uso da SDK.

Este tópico fornece informações que ajudarão você a começar a usar o Message Processing Framework. Além das informações de pré-requisitos e configuração, é fornecido um tutorial que mostra como implementar um cenário comum.

Pré-requisitos e configuração

- As credenciais que você fornece para seu aplicativo devem ter as permissões apropriadas para o serviço de mensagens e as operações que ele usa. Para obter mais informações, consulte os tópicos de segurança para SQS, SNS e EventBridgeem seus respectivos guias do desenvolvedor. Veja também a parte do arquivo README GitHub que discute permissões específicas.
- Para usar o AWS Message Processing Framework para.NET, você deve adicionar o `AWS.Messaging` NuGetpacote ao seu projeto. Por exemplo:

```
dotnet add package AWS.Messaging
```

- A estrutura se integra ao contêiner de serviço de injeção de dependência (DI) do.NET. Você pode configurar a estrutura durante a inicialização do seu aplicativo ligando `AddAWSMessageBus` para adicioná-la ao contêiner de DI.

```
var builder = WebApplication.CreateBuilder(args);

// Register the AWS Message Processing Framework for .NET
builder.Services.AddAWSMessageBus(builder =>
{
    // Register that you'll publish messages of type ChatMessage to an existing queue
    builder.AddSQSPublisher<ChatMessage>("https://sqs.us-
west-2.amazonaws.com/012345678910/MyAppProd");
});
```

Tutorial

Este tutorial demonstra como usar o AWS Message Processing Framework para.NET. Ele cria dois aplicativos: uma API ASP.NET Core Minimal que envia mensagens para uma fila do Amazon

SQS quando recebe uma solicitação em um endpoint da API e um aplicativo de console de longa execução que pesquisa essas mensagens e as processa.

- As instruções neste tutorial favorecem a CLI do.NET, mas você pode realizar esse tutorial usando ferramentas multiplataforma, como a CLI do.NET ou o Microsoft Visual Studio. Para obter informações sobre ferramentas, consulteInstalando e configurando seu conjunto de ferramentas para o AWS SDK para .NET.
- Este tutorial pressupõe que você esteja usando seu `[default]` perfil para obter credenciais. Também pressupõe que credenciais de curto prazo estejam disponíveis com as permissões apropriadas para enviar e receber mensagens do Amazon SQS. Para obter mais informações, consulte Autenticando o com AWS SDK para .NET AWS os tópicos de segurança do SQS.

> **Note**
>
> Ao executar este tutorial, você pode incorrer em custos com mensagens SQS.

Etapas

- Criar uma fila SQS
- Crie e execute o aplicativo de publicação
- Crie e execute o aplicativo de manipulação
- Limpeza

Criar uma fila SQS

Este tutorial requer uma fila SQS para enviar e receber mensagens. Uma fila pode ser criada usando um dos seguintes comandos para o AWS CLI ou o. Ferramentas da AWS para PowerShell Anote a URL da fila que é retornada para que você possa especificá-la na configuração da estrutura a seguir.

AWS CLI

```
aws sqs create-queue --queue-name DemoQueue
```

Ferramentas da AWS para PowerShell

```
New-SQSQueue -QueueName DemoQueue
```

Crie e execute o aplicativo de publicação

Use o procedimento a seguir para criar e executar o aplicativo de publicação.

1. Abra um prompt de comando ou terminal. Localize ou crie uma pasta do sistema operacional na qual você pode criar um projeto .NET.
2. Nessa pasta, execute o comando a seguir para criar o projeto .NET.

```
dotnet new webapi --name Publisher
```

3. Navegue até a pasta do novo projeto. Adicione uma dependência na estrutura de processamento de AWS mensagens para.NET.

```
cd Publisher
dotnet add package AWS.Messaging
```

> **Note**
>
> Se você estiver usando Centro de Identidade do AWS IAM para autenticação, não se esqueça de também adicionar `AWSSDK.SSO` `AWSSDK.SSOOIDC` e.

4. Substitua o código em `Program.cs` pelo código a seguir.

```
using AWS.Messaging;
using Microsoft.AspNetCore.Mvc;
using Publisher;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/
swashbuckle.
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();


// Configure the AWS Message Processing Framework for .NET.
builder.Services.AddAWSMessageBus(builder =>
{
    // Check for input SQS URL.
```

```
    // The SQS URL should be passed as a command line argument or set in the Debug
 launch profile.
    if ((args.Length == 1) && (args[0].Contains("https://sqs.")))
    {
        // Register that you'll publish messages of type GreetingMessage:
        // 1. To a specified queue.
        // 2. Using the message identifier "greetingMessage", which will be used
        //    by handlers to route the message to the appropriate handler.
        builder.AddSQSPublisher<GreetingMessage>(args[0], "greetingMessage");
    }
    // You can map additional message types to queues or topics here as well.
});
var app = builder.Build();


// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseHttpsRedirection();

// Create an API Endpoint that receives GreetingMessage objects
// from the caller and then sends them as an SQS message.
app.MapPost("/greeting", async ([FromServices] IMessagePublisher publisher,
 Publisher.GreetingMessage message) =>
    {
        return await PostGreeting(message, publisher);
    })
.WithName("SendGreeting")
.WithOpenApi();

app.Run();

public partial class Program
{
    /// <summary>
    /// Endpoint for posting a greeting message.
    /// </summary>
    /// <param name="greetingMessage">The greeting message.</param>
    /// <param name="messagePublisher">The message publisher.</param>
    /// <returns>Async task result.</returns>
```

```
    public static async Task<IResult> PostGreeting(GreetingMessage greetingMessage,
        IMessagePublisher messagePublisher)
    {
        if (greetingMessage.SenderName == null || greetingMessage.Greeting == null)
        {
            return Results.BadRequest();
        }

        // Publish the message to the queue configured above.
        await messagePublisher.PublishAsync(greetingMessage);

        return Results.Ok();
    }
}

namespace Publisher
{
    /// <summary>
    /// This class represents the message contents.
    /// </summary>
    public class GreetingMessage
    {
        public string? SenderName { get; set; }
        public string? Greeting { get; set; }
    }
}
```

5. Execute o seguinte comando: Isso deve abrir uma janela do navegador com a interface do usuário do Swagger, que permite explorar e testar sua API.

```
dotnet watch run <queue URL created earlier>
```

6. Abra o `/greeting` endpoint e escolha Testar.
7. Especifique `senderName` os `greeting` valores da mensagem e escolha Executar. Isso invoca sua API, que envia a mensagem SQS.

Crie e execute o aplicativo de manipulação

Use o procedimento a seguir para criar e executar o aplicativo de manipulação.

1. Abra um prompt de comando ou terminal. Localize ou crie uma pasta do sistema operacional na qual você pode criar um projeto .NET.

2. Nessa pasta, execute o comando a seguir para criar o projeto .NET.

```
dotnet new console --name Handler
```

3. Navegue até a pasta do novo projeto. Adicione uma dependência na estrutura de processamento de AWS mensagens para.NET. Adicione também o `Microsoft.Extensions.Hosting` pacote, que permite configurar a estrutura por meio do host genérico do.NET.

```
cd Handler
dotnet add package AWS.Messaging
dotnet add package Microsoft.Extensions.Hosting
```

> **Note**
>
> Se você estiver usando Centro de Identidade do AWS IAM para autenticação, não se esqueça de também adicionar `AWSSDK.SSO` `AWSSDK.SSOOIDC` e.

4. Substitua o código em `Program.cs` pelo código a seguir.

```
using AWS.Messaging;
using Handler;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

var builder = Host.CreateDefaultBuilder(args);

builder.ConfigureServices(services =>
{
    // Register the AWS Message Processing Framework for .NET.
    services.AddAWSMessageBus(builder =>
    {
        // Check for input SQS URL.
        // The SQS URL should be passed as a command line argument or set in the
 Debug launch profile.
        if ((args.Length == 1) && (args[0].Contains("https://sqs.")))
        {
            // Register you'll poll the following queue.
            builder.AddSQSPoller(args[0]);
```

```
            // And that messages of type "greetingMessage" should be:
            // 1. Deserialized as GreetingMessage objects.
            // 2. Which are then passed to GreetingMessageHandler.
            builder.AddMessageHandler<GreetingMessageHandler,
 GreetingMessage>("greetingMessage");

        }
        // You can add additional message handlers here, using different message
 types.
    });
});

var host = builder.Build();
await host.RunAsync();

namespace Handler
{
    /// <summary>
    /// This class represents the message contents.
    /// </summary>
    public class GreetingMessage
    {
        public string? SenderName { get; set; }
        public string? Greeting { get; set; }
    }

    /// <summary>
    /// This handler is invoked each time you receive the message.
    /// </summary>
    public class GreetingMessageHandler : IMessageHandler<GreetingMessage>
    {
        public Task<MessageProcessStatus> HandleAsync(
            MessageEnvelope<GreetingMessage> messageEnvelope,
            CancellationToken token = default)
        {
            Console.WriteLine(
                $"Received message {messageEnvelope.Message.Greeting} from
 {messageEnvelope.Message.SenderName}");
            return Task.FromResult(MessageProcessStatus.Success());
        }
    }
}
```

5. Execute o seguinte comando: Isso inicia uma pesquisa de longa duração.

```
dotnet run <queue URL created earlier>
```

Logo após a inicialização, o aplicativo receberá a mensagem que foi enviada na primeira parte deste tutorial e registrará a seguinte mensagem:

```
Received message {greeting} from {senderName}
```

6. Pressione `Ctrl+C` para parar o poller.

Limpeza

Use um dos comandos a seguir para o AWS CLI ou Ferramentas da AWS para PowerShell para excluir a fila.

AWS CLI

```
aws sqs delete-queue --queue-url "<queue URL created earlier>"
```

Ferramentas da AWS para PowerShell

```
Remove-SQSQueue -QueueUrl "<queue URL created earlier>"
```

Publique mensagens com o AWS Message Processing Framework para.NET

O AWS Message Processing Framework para.NET oferece suporte à publicação de um ou mais tipos de mensagens, ao processamento de um ou mais tipos de mensagens ou à realização de ambos no mesmo aplicativo.

O código a seguir mostra a configuração de um aplicativo que está publicando diferentes tipos de mensagens em diferentes AWS serviços.

```
var builder = WebApplication.CreateBuilder(args);

// Register the AWS Message Processing Framework for .NET
builder.Services.AddAWSMessageBus(builder =>
{
    // Register that you'll send messages of type ChatMessage to an existing queue
```

```
    builder.AddSQSPublisher<ChatMessage>("https://sqs.us-
west-2.amazonaws.com/012345678910/MyAppProd");

    // Register that you'll publish messages of type OrderInfo to an existing SNS topic
    builder.AddSNSPublisher<OrderInfo>("arn:aws:sns:us-west-2:012345678910:MyAppProd");

    // Register that you'll publish messages of type FoodItem to an existing
 EventBridge bus
    builder.AddEventBridgePublisher<FoodItem>("arn:aws:events:us-
west-2:012345678910:event-bus/default");
});
```

Depois de registrar a estrutura durante a inicialização, injete o genérico `IMessagePublisher` em seu código. Chame seu `PublishAsync` método para publicar qualquer um dos tipos de mensagem que foram configurados acima. O editor genérico determinará o destino para o qual rotear a mensagem com base em seu tipo.

No exemplo a seguir, um controlador ASP.NET MVC recebe `ChatMessage` mensagens e `OrderInfo` eventos dos usuários e os publica no Amazon SQS e no Amazon SNS, respectivamente. Ambos os tipos de mensagem podem ser publicados usando o editor genérico que foi configurado acima.

```
[ApiController]
[Route("[controller]")]
public class PublisherController : ControllerBase
{
    private readonly IMessagePublisher _messagePublisher;

    public PublisherController(IMessagePublisher messagePublisher)
    {
        _messagePublisher = messagePublisher;
    }

    [HttpPost("chatmessage", Name = "Chat Message")]
    public async Task<IActionResult> PublishChatMessage([FromBody] ChatMessage message)
    {
        // Perform business and validation logic on the ChatMessage here.
        if (message == null)
        {
            return BadRequest("A chat message was not submitted. Unable to forward to
 the message queue.");
        }
```

```
        if (string.IsNullOrEmpty(message.MessageDescription))
        {
            return BadRequest("The MessageDescription cannot be null or empty.");
        }

        // Send the ChatMessage to SQS, using the generic publisher.
        await _messagePublisher.PublishAsync(message);

        return Ok();
    }

    [HttpPost("order", Name = "Order")]
    public async Task<IActionResult> PublishOrder([FromBody] OrderInfo message)
    {
        if (message == null)
        {
            return BadRequest("An order was not submitted.");
        }

        // Publish the OrderInfo to SNS, using the generic publisher.
        await _messagePublisher.PublishAsync(message);

        return Ok();
    }
}
```

Para rotear uma mensagem para a lógica de tratamento apropriada, a estrutura usa metadados chamados identificador do tipo de mensagem. Por padrão, esse é o nome completo do tipo.NET da mensagem, incluindo seu nome de montagem. Se você estiver enviando e manipulando mensagens, esse mecanismo funciona bem se você compartilhar a definição dos objetos de mensagem entre projetos. No entanto, se as mensagens forem redefinidas em namespaces diferentes ou se você estiver trocando mensagens com outras estruturas ou linguagens de programação, talvez seja necessário substituir o identificador do tipo de mensagem.

```
var builder = Host.CreateDefaultBuilder(args);

builder.ConfigureServices(services =>
{
    // Register the AWS Message Processing Framework for .NET
    services.AddAWSMessageBus(builder =>
    {
```

```
        // Register that you'll publish messages of type GreetingMessage to an existing
 queue
        builder.AddSQSPublisher<GreetingMessage>("https://sqs.us-
west-2.amazonaws.com/012345678910/MyAppProd", "greetingMessage");
    });
});
```

Editores específicos de serviços

O exemplo mostrado acima usa o genéricoIMessagePublisher, que pode ser publicado em qualquer AWS serviço compatível com base no tipo de mensagem configurado. A estrutura também fornece editores específicos de serviços para Amazon SQS, Amazon SNS e Amazon. EventBridge Esses editores específicos expõem opções que se aplicam somente a esse serviço e podem ser injetadas usando os tiposISQSPublisher, e. ISNSPublisher IEventBridgePublisher

Por exemplo, ao enviar mensagens para uma fila FIFO do SQS, você deve definir o ID do grupo de mensagens apropriado. O código a seguir mostra o ChatMessage exemplo novamente, mas agora usando an ISQSPublisher para definir opções específicas do SQS.

```
public class PublisherController : ControllerBase
{
    private readonly ISQSPublisher _sqsPublisher;

    public PublisherController(ISQSPublisher sqsPublisher)
    {
        _sqsPublisher = sqsPublisher;
    }

    [HttpPost("chatmessage", Name = "Chat Message")]
    public async Task<IActionResult> PublishChatMessage([FromBody] ChatMessage message)
    {
        // Perform business and validation logic on the ChatMessage here
        if (message == null)
        {
            return BadRequest("A chat message was not submitted. Unable to forward to
 the message queue.");
        }
        if (string.IsNullOrEmpty(message.MessageDescription))
        {
            return BadRequest("The MessageDescription cannot be null or empty.");
        }
```

```
        // Send the ChatMessage to SQS using the injected ISQSPublisher, with SQS-
specific options
        await _sqsPublisher.SendAsync(message, new SQSOptions
        {
            DelaySeconds = <delay-in-seconds>,
            MessageAttributes = <message-attributes>,
            MessageDeduplicationId = <message-deduplication-id>,
            MessageGroupId = <message-group-id>
        });

        return Ok();
    }
}
```

O mesmo pode ser feito para o SNS e EventBridge, usando `ISNSPublisher` e, `IEventBridgePublisher` respectivamente.

```
await _snsPublisher.PublishAsync(message, new SNSOptions
{
    Subject = <subject>,
    MessageAttributes = <message-attributes>,
    MessageDeduplicationId = <message-deduplication-id>,
    MessageGroupId = <message-group-id>
});
```

```
await _eventBridgePublisher.PublishAsync(message, new EventBridgeOptions
{
    DetailType = <detail-type>,
    Resources = <resources>,
    Source = <source>,
    Time = <time>,
    TraceHeader = <trace-header>
});
```

Por padrão, as mensagens de um determinado tipo são enviadas para o destino configurado com antecedência. No entanto, você pode substituir o destino de uma única mensagem usando os editores específicos da mensagem. Você também pode substituir o AWS SDK para .NET cliente subjacente usado para publicar a mensagem, o que pode ser útil em aplicativos multilocatários nos quais você precisa alterar funções ou credenciais, dependendo do destino.

```
await _sqsPublisher.SendAsync(message, new SQSOptions
```

```
{
    OverrideClient = <override IAmazonSQS client>,
    QueueUrl = <override queue URL>
});
```

Consuma mensagens com o AWS Message Processing Framework para.NET

A Estrutura de Processamento de AWS Mensagens para.NET permite que você consuma mensagens que foram publicadas usando a estrutura ou um dos serviços de mensagens. As mensagens podem ser consumidas de várias maneiras, algumas das quais estão descritas abaixo.

Manipuladores de mensagens

Para consumir mensagens, implemente um manipulador de mensagens usando a `IMessageHandler` interface para cada tipo de mensagem que você deseja processar. O mapeamento entre tipos de mensagens e manipuladores de mensagens é configurado na inicialização do projeto.

```
await Host.CreateDefaultBuilder(args)
    .ConfigureServices(services =>
    {
        // Register the AWS Message Processing Framework for .NET
        services.AddAWSMessageBus(builder =>
        {
            // Register an SQS Queue that the framework will poll for messages.
            // NOTE: The URL given below is an example. Use the appropriate URL for
 your SQS Queue.
            builder.AddSQSPoller("https://sqs.us-west-2.amazonaws.com/012345678910/
MyAppProd");

            // Register all IMessageHandler implementations with the message type they
 should process.
            // Here messages that match our ChatMessage .NET type will be handled by
 our ChatMessageHandler
            builder.AddMessageHandler<ChatMessageHandler, ChatMessage>();
        });
    })
    .Build()
    .RunAsync();
```

O código a seguir mostra um exemplo de manipulador de mensagens para uma `ChatMessage` mensagem.

```
public class ChatMessageHandler : IMessageHandler<ChatMessage>
{
    public Task<MessageProcessStatus> HandleAsync(MessageEnvelope<ChatMessage>
 messageEnvelope, CancellationToken token = default)
    {
        // Add business and validation logic here.
        if (messageEnvelope == null)
        {
            return Task.FromResult(MessageProcessStatus.Failed());
        }

        if (messageEnvelope.Message == null)
        {
            return Task.FromResult(MessageProcessStatus.Failed());
        }

        ChatMessage message = messageEnvelope.Message;

        Console.WriteLine($"Message Description: {message.MessageDescription}");

        // Return success so the framework will delete the message from the queue.
        return Task.FromResult(MessageProcessStatus.Success());
    }
}
```

O externo `MessageEnvelope` contém metadados usados pela estrutura. Sua `message` propriedade é o tipo de mensagem (nesse caso`ChatMessage`).

Você pode retornar `MessageProcessStatus.Success()` para indicar que a mensagem foi processada com sucesso e que a estrutura excluirá a mensagem da fila do Amazon SQS. Ao retornar`MessageProcessStatus.Failed()`, a mensagem permanecerá na fila, onde poderá ser processada novamente ou movida para uma fila de mensagens mortas, se configurada.

Tratamento de mensagens em um processo de longa duração

Você pode chamar `AddSQSPoller` com uma URL de fila SQS para iniciar uma longa execução `BackgroundService`que pesquisará continuamente a fila e processará mensagens.

```
await Host.CreateDefaultBuilder(args)
    .ConfigureServices(services =>
    {
        // Register the AWS Message Processing Framework for .NET
```

```
        services.AddAWSMessageBus(builder =>
        {
            // Register an SQS Queue that the framework will poll for messages.
            // NOTE: The URL given below is an example. Use the appropriate URL for
 your SQS Queue.
            builder.AddSQSPoller("https://sqs.us-west-2.amazonaws.com/012345678910/
MyAppProd", options =>
            {
                // The maximum number of messages from this queue that the framework
 will process concurrently on this client.
                options.MaxNumberOfConcurrentMessages = 10;

                // The duration each call to SQS will wait for new messages.
                options.WaitTimeSeconds = 20;
            });

            // Register all IMessageHandler implementations with the message type they
 should process.
            builder.AddMessageHandler<ChatMessageHandler, ChatMessage>();
        });
    })
    .Build()
    .RunAsync();
```

Configurando o SQS Message Poller

O poller de mensagens do SQS pode ser configurado pelo `SQSMessagePollerOptions` durante a chamada. `AddSQSPoller`

- `MaxNumberOfConcurrentMessages`- O número máximo de mensagens da fila a serem processadas simultaneamente. O valor padrão é 10.
- `WaitTimeSeconds`- A duração (em segundos) pela qual a chamada do `ReceiveMessage` SQS espera que uma mensagem chegue na fila antes de retornar. Se uma mensagem estiver disponível, a chamada retornará antes de`WaitTimeSeconds`. O valor padrão é 20.

Tratamento do tempo limite de visibilidade da mensagem

As mensagens do SQS têm um período de tempo limite de visibilidade. Quando um consumidor começa a lidar com uma determinada mensagem, ela permanece na fila, mas fica oculta dos outros consumidores para evitar processá-la mais de uma vez. Se a mensagem não for tratada e

excluída antes de se tornar visível novamente, outro consumidor poderá tentar lidar com a mesma mensagem.

A estrutura rastreará e tentará estender o tempo limite de visibilidade das mensagens que está manipulando atualmente. Você pode configurar esse comportamento no `SQSMessagePollerOptions` ao ligar`AddSQSPoller`.

- `VisibilityTimeout`- A duração em segundos em que as mensagens recebidas são ocultadas das solicitações de recuperação subsequentes. O valor padrão é 30.
- `VisibilityTimeoutExtensionThreshold`- Quando o tempo limite de visibilidade de uma mensagem estiver dentro de alguns segundos após a expiração, a estrutura estenderá o tempo limite de visibilidade (em mais `VisibilityTimeout` alguns segundos). O valor padrão é 5.
- `VisibilityTimeoutExtensionHeartbeatInterval`- Com que frequência, em segundos, a estrutura verificará as mensagens que estão prestes a expirar e, em `VisibilityTimeoutExtensionThreshold` seguida, estenderá o tempo limite de visibilidade. O valor padrão é 1.

No exemplo a seguir, a estrutura verificará a cada 1 segundo as mensagens que ainda estão sendo tratadas. Para essas mensagens dentro de 5 segundos após se tornarem visíveis novamente, a estrutura estenderá automaticamente o tempo limite de visibilidade de cada mensagem em mais 30 segundos.

```
// NOTE: The URL given below is an example. Use the appropriate URL for your SQS Queue.
builder.AddSQSPoller("https://sqs.us-west-2.amazonaws.com/012345678910/MyAppProd",
 options =>
{
    options.VisibilityTimeout = 30;
    options.VisibilityTimeoutExtensionThreshold = 5;
    VisibilityTimeoutExtensionHeartbeatInterval = 1;
});
```

Manipulação de mensagens em AWS Lambda funções

Você pode usar o AWS Message Processing Framework para.NET com a integração do SQS com o Lambda. Isso é fornecido pelo `AWS.Messaging.Lambda` pacote. Consulte seu README para começar.

Usando o FIFO com a estrutura de processamento de AWS mensagens para.NET

Para casos de uso em que a ordenação e a desduplicação de mensagens são essenciais, o AWS Message Processing Framework para.NET oferece suporte first-in-first-out (FIFO) às filas do Amazon SQS e aos tópicos do Amazon SNS.

Publicação

Ao publicar mensagens em uma fila ou tópico FIFO, você deve definir a ID do grupo de mensagens, que especifica o grupo ao qual a mensagem pertence. As mensagens dentro de um grupo são processadas em ordem. Você pode definir isso nos editores de mensagens específicos do SQS e do SNS.

```
await _sqsPublisher.PublishAsync(message, new SQSOptions
{
    MessageDeduplicationId = <message-deduplication-id>,
    MessageGroupId = <message-group-id>
});
```

Assinatura

Ao lidar com mensagens de uma fila FIFO, a estrutura manipula as mensagens dentro de um determinado grupo de mensagens na ordem em que foram recebidas para cada `ReceiveMessages` chamada. A estrutura entra nesse modo de operação automaticamente quando configurada com uma fila terminando em `.fifo`.

```
await Host.CreateDefaultBuilder(args)
    .ConfigureServices(services =>
    {
        // Register the AWS Message Processing Framework for .NET.
        services.AddAWSMessageBus(builder =>
        {
            // Because this is a FIFO queue, the framework automatically handles these
 messages in order.
            builder.AddSQSPoller("https://sqs.us-west-2.amazonaws.com/012345678910/
MPF.fifo");
            builder.AddMessageHandler<OrderMessageHandler, OrderMessage>();
        });
    })
    .Build()
    .RunAsync();
```

Registro e telemetria aberta para a estrutura de processamento de AWS mensagens para.NET

A Estrutura de Processamento de AWS Mensagens para.NET é instrumentada OpenTelemetry para registrar rastreamentos para cada mensagem publicada ou tratada pela estrutura. Isso é fornecido pelo `AWS.Messaging.Telemetry.OpenTelemetry`pacote. Consulte seu README para começar.

> **Note**
>
> Para obter informações de segurança relacionadas ao registro, consulteSegurança para a estrutura de processamento de AWS mensagens para.NET.

Personalize a estrutura de processamento de AWS mensagens para.NET

O AWS Message Processing Framework para.NET cria, envia e manipula mensagens em três “camadas” diferentes:

1. Na camada mais externa, a estrutura cria a solicitação ou resposta AWS-nativa específica para um serviço. Com o Amazon SQS, por exemplo, ele cria `SendMessage`solicitações e trabalha com os `Message`objetos definidos pelo serviço.
2. Dentro da solicitação e resposta do SQS, a estrutura define o `MessageBody` elemento (ou `Message` para o Amazon SNS `Detail` ou para a EventBridge Amazon) como um formato JSON. CloudEvent Ele contém metadados definidos pela estrutura que podem ser acessados no `MessageEnvelope` objeto ao manipular uma mensagem.
3. Na camada mais interna, o `data` atributo dentro do objeto CloudEvent JSON contém uma serialização JSON do objeto.NET que foi enviado ou recebido como mensagem.

```
{
    "id":"b02f156b-0f02-48cf-ae54-4fbbe05cffba",
    "source":"/aws/messaging",
    "specversion":"1.0",
    "type":"Publisher.Models.ChatMessage",
    "time":"2023-11-21T16:36:02.8957126+00:00",
    "data":"<the ChatMessage object serialized as JSON>"
}
```

Você pode personalizar a forma como o envelope da mensagem é configurado e lido:

- `"id"`identifica a mensagem de forma exclusiva. Por padrão, ele é definido como um novo GUID, mas isso pode ser substituído implementando o seu próprio `IMessageIdGenerator` e injetando-o no contêiner de DI.
- `"type"`controla como a mensagem é roteada para os manipuladores. Por padrão, isso usa o nome completo do tipo.NET que corresponde à mensagem. Você pode substituir isso por meio do `messageTypeIdentifier` parâmetro ao mapear o tipo de mensagem para o destino via`AddSQSPublisher`,`AddSNSPublisher`, ou`AddEventBridgePublisher`.
- `"source"`indica qual sistema ou servidor enviou a mensagem.
  - Esse será o nome da função se estiver publicando AWS Lambda, o nome do cluster e o ARN da tarefa se estiver no Amazon ECS, o ID da instância se estiver na Amazon, caso contrário EC2, um valor alternativo de. `/aws/messaging`
  - Você pode substituir isso por meio de `AddMessageSource` ou `AddMessageSourceSuffix` no`MessageBusBuilder`.
- `"time"`definido para o atual DateTime em UTC. Isso pode ser substituído implementando o seu próprio `IDateTimeHandler` e injetando-o no contêiner DI.
- `"data"`contém uma representação JSON do objeto.NET que foi enviado ou recebido como mensagem:
  - `ConfigureSerializationOptions`on `MessageBusBuilder` permite que você configure o `System.Text.Json.JsonSerializerOptions`que será usado ao serializar e desserializar a mensagem.
  - Para injetar atributos adicionais ou transformar o envelope da mensagem depois que a estrutura o cria, você pode implementar `ISerializationCallback` e registrar isso por meio de on. `AddSerializationCallback` `MessageBusBuilder`

Segurança para a estrutura de processamento de AWS mensagens para.NET

A Estrutura de Processamento de AWS Mensagens para.NET depende do AWS SDK para .NET para se comunicar com. AWS Para obter mais informações sobre segurança no AWS SDK para .NET, consulteSegurança para este AWS produto ou serviço.

Por motivos de segurança, a estrutura não registra mensagens de dados enviadas pelo usuário. Se quiser habilitar essa funcionalidade para fins de depuração, você precisa chamar o Message Bus da seguinte `EnableDataMessageLogging()` forma:

```
builder.Services.AddAWSMessageBus(bus =>
```

```
{
    builder.EnableDataMessageLogging();
});
```

Se você descobrir um possível problema de segurança, consulte a política de segurança para relatar informações.

Integração AWS com o.NET Aspire no AWS SDK para .NET

O.NET Aspire é uma nova forma de criar aplicativos prontos para a nuvem. Em particular, ele fornece uma orquestração para ambientes locais nos quais executar, conectar e depurar os componentes de aplicativos distribuídos. Para melhorar o ciclo interno de desenvolvimento de aplicativos prontos para a nuvem, foram criadas integrações com o.NET Aspire para conectar seus aplicativos.NET aos recursos. AWS Essas integrações estão disponíveis por meio do pacote Aspire.Hosting.AWS NuGet .

As seguintes integrações do.NET Aspire estão disponíveis:

- A capacidade de provisionar seus AWS recursos por meio de CloudFormation. Essa integração é utilizada no projeto.NET Aspire AppHost .

  Para obter mais informações, consulte a postagem do blog Integrando AWS com o.NET Aspire.
- Instalando, configurando e conectando o AWS SDK para .NET ao Amazon DynamoDB local. Essa integração é utilizada no projeto.NET Aspire AppHost .

  Para obter mais informações, consulte a postagem do blog Integrando AWS com o.NET Aspire.
- Habilite um ambiente de desenvolvimento local para AWS Lambdafunções. Essa integração é utilizada no projeto.NET Aspire AppHost .

  Para obter mais informações, consulte as postagens do blog Criando e depurando aplicativos.NET Lambda com o.NET Aspire (Parte 1) e Criando e depurando aplicativos.NET Lambda com o.NET Aspire (Parte 2).

> **Note**
>
> Esta é uma documentação de pré-lançamento de um recurso em versão de pré-visualização. Está sujeita a alteração.

Como esse recurso está em pré-visualização, você precisará optar pelos recursos de pré-visualização. Para obter informações adicionais sobre esse recurso de visualização e como se inscrever, consulte o problema do rastreador de desenvolvimento em. GitHub

Mais informações

Para obter informações adicionais e detalhes sobre como você pode usar as integrações que estão disponíveis no Aspire.Hosting.AWS, consulte os seguintes recursos.

- Postagem no blog Integração AWS com o.NET Aspire.
- Publicações no blog Criação e depuração de aplicativos.NET Lambda com o.NET Aspire (Parte 1) e Criação e depuração de aplicativos.NET Lambda com o.NET Aspire (Parte 2).
- O integrations-on-dotnet-aspirerepositório -for-aws ativado. GitHub
- O README detalhado do pacote Aspire.Hosting.AWS NuGet .

Support para outros AWS serviços e configuração

Os AWS serviços de AWS SDK para .NET suporte, além dos descritos nas seções anteriores. Para obter informações sobre todos os serviços compatíveis, consulte a Referência AWS SDK para .NET da API. APIs

Além dos namespaces para AWS serviços individuais, o AWS SDK para .NET também fornece o seguinte: APIs

Área	Descrição	Recursos
AWS Support	Acesso programático aos casos de AWS suporte e aos recursos do Trusted Advisor.	Veja a Amazon. AWSSupport e Amazon. AWSSupport.Modelo.
Geral	Classes auxiliares e enumerações.	Consulte Amazon e Amazon.Util.

SDK para .NET (v4) exemplos de código

Os exemplos de código neste tópico mostram como usar o AWS SDK para .NET (v4) com AWS.

As noções básicas são exemplos de código que mostram como realizar as operações essenciais em um serviço.

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar perfis de serviço individuais, você pode ver as ações no contexto em seus cenários relacionados.

Cenários são exemplos de código que mostram como realizar tarefas específicas chamando várias funções dentro de um serviço ou combinadas com outros Serviços da AWS.

Alguns serviços contêm categorias de exemplo adicionais que mostram como utilizar bibliotecas ou funções específicas do serviço.

Services

- Exemplos de Aurora usando SDK para .NET (v4)
- Exemplos de Auto Scaling usando SDK para .NET (v4)
- Exemplos do Amazon Bedrock usando SDK para .NET (v4)
- Exemplos de Amazon Bedrock Runtime usando SDK para .NET (v4)
- CloudFormation exemplos usando SDK para .NET (v4)
- CloudWatch exemplos usando SDK para .NET (v4)
- CloudWatch Exemplos de registros usando SDK para .NET (v4)
- Exemplos de provedores de identidade Amazon Cognito usando SDK para .NET (v4)
- AWS Control Tower exemplos usando SDK para .NET (v4)
- Exemplos SDK para .NET do DynamoDB usando (v4)
- Exemplos do Amazon EC2 usando SDK para .NET (v4)
- Exemplos do Amazon ECS usando SDK para .NET (v4)
- AWS IoT exemplos usando SDK para .NET (v4)
- AWS IoT data exemplos usando SDK para .NET (v4)
- Exemplos do Amazon Redshift usando SDK para .NET (v4)
- Exemplos do Amazon S3 usando SDK para .NET (v4)

Exemplos de Aurora usando SDK para .NET (v4)

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET (v4) com o Aurora.

As noções básicas são exemplos de código que mostram como realizar as operações essenciais em um serviço.

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar perfis de serviço individuais, você pode ver as ações no contexto em seus cenários relacionados.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- Conceitos básicos
- Conceitos básicos
- Ações

Conceitos básicos

Olá, Aurora

O exemplo de código a seguir mostra como começar a usar o Aurora.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
using Amazon.RDS;
using Amazon.RDS.Model;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
```

```
namespace AuroraActions;

public static class HelloAurora
{
    static async Task Main(string[] args)
    {
        // Use the AWS .NET Core Setup package to set up dependency injection for
 the
        // Amazon Relational Database Service (Amazon RDS).
        // Use your AWS profile name, or leave it blank to use the default profile.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonRDS>()
            ).Build();

        // Now the client is available for injection. Fetching it directly here for
 example purposes only.
        var rdsClient = host.Services.GetRequiredService<IAmazonRDS>();

        // You can use await and any of the async methods to get a response.
        var response = await rdsClient.DescribeDBClustersAsync(new
 DescribeDBClustersRequest { IncludeShared = true });
        Console.WriteLine($"Hello Amazon RDS Aurora! Let's list some clusters in
 this account:");
        if (response.DBClusters == null)
        {
            Console.WriteLine($"\tNo clusters found.");
        }
        else
        {
            foreach (var cluster in response.DBClusters)
            {
                Console.WriteLine(
                    $"\tCluster: database: {cluster.DatabaseName} identifier:
 {cluster.DBClusterIdentifier}.");
            }
        }
    }
}
```

- Para obter detalhes da API, consulte Descrever DBClusters na Referência AWS SDK para .NET da API.

Conceitos básicos

Conheça os conceitos básicos

O exemplo de código a seguir mostra como:

- Criar um grupo de parâmetros de cluster do banco de dados do Aurora e definir os valores dos parâmetros.
- Criar um cluster de banco de dados que use o grupo de parâmetros.
- Criar uma instância de banco de dados que contenha um banco de dados.
- Crie um snapshot do cluster do banco de dados e limpe os recursos.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Execute um cenário interativo em um prompt de comando.

```
using Amazon.RDS;
using Amazon.RDS.Model;
using AuroraActions;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Logging.Console;
using Microsoft.Extensions.Logging.Debug;

namespace AuroraScenario;

/// <summary>
/// Scenario for Amazon Aurora examples.
```

```
/// </summary>
public class AuroraScenario
{
    /*
    Before running this .NET code example, set up your development environment,
 including your credentials.

    This .NET example performs the following tasks:
    1.  Return a list of the available DB engine families for Aurora MySql using the
 DescribeDBEngineVersionsAsync method.
    2.  Select an engine family and create a custom DB cluster parameter group using
 the CreateDBClusterParameterGroupAsync method.
    3.  Get the parameter group using the DescribeDBClusterParameterGroupsAsync
 method.
    4.  Get some parameters in the group using the DescribeDBClusterParametersAsync
 method.
    5.  Parse and display some parameters in the group.
    6.  Modify the auto_increment_offset and auto_increment_increment parameters
        using the ModifyDBClusterParameterGroupAsync method.
    7.  Get and display the updated parameters using the
 DescribeDBClusterParametersAsync method with a source of "user".
    8.  Get a list of allowed engine versions using the
 DescribeDBEngineVersionsAsync method.
    9.  Create an Aurora DB cluster that contains a MySql database and uses the
 parameter group.
        using the CreateDBClusterAsync method.
    10. Wait for the DB cluster to be ready using the DescribeDBClustersAsync
 method.
    11. Display and select from a list of instance classes available for the
 selected engine and version
        using the paginated DescribeOrderableDBInstanceOptions method.
    12. Create a database instance in the cluster using the CreateDBInstanceAsync
 method.
    13. Wait for the DB instance to be ready using the DescribeDBInstances method.
    14. Display the connection endpoint string for the new DB cluster.
    15. Create a snapshot of the DB cluster using the CreateDBClusterSnapshotAsync
 method.
    16. Wait for DB snapshot to be ready using the DescribeDBClusterSnapshotsAsync
 method.
    17. Delete the DB instance using the DeleteDBInstanceAsync method.
    18. Delete the DB cluster using the DeleteDBClusterAsync method.
    19. Wait for DB cluster to be deleted using the DescribeDBClustersAsync methods.
    20. Delete the cluster parameter group using the
 DeleteDBClusterParameterGroupAsync.
```

```
   */

   private static readonly string sepBar = new('-', 80);
   private static AuroraWrapper auroraWrapper = null!;
   private static ILogger logger = null!;
   private static readonly string engine = "aurora-mysql";
   static async Task Main(string[] args)
   {
       // Set up dependency injection for the Amazon Relational Database Service
(Amazon RDS).
       using var host = Host.CreateDefaultBuilder(args)
           .ConfigureLogging(logging =>
               logging.AddFilter("System", LogLevel.Debug)
                   .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                   .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
           .ConfigureServices((_, services) =>
               services.AddAWSService<IAmazonRDS>()
                   .AddTransient<AuroraWrapper>()
           )
           .Build();

       logger = LoggerFactory.Create(builder =>
       {
           builder.AddConsole();
       }).CreateLogger<AuroraScenario>();

       auroraWrapper = host.Services.GetRequiredService<AuroraWrapper>();

       Console.WriteLine(sepBar);
       Console.WriteLine(
           "Welcome to the Amazon Aurora: get started with DB clusters example.");
       Console.WriteLine(sepBar);

       DBClusterParameterGroup parameterGroup = null!;
       DBCluster? newCluster = null;
       DBInstance? newInstance = null;

       try
       {
           var parameterGroupFamily = await ChooseParameterGroupFamilyAsync();

           parameterGroup = await
CreateDBParameterGroupAsync(parameterGroupFamily);
```

```
            var parameters = await
DescribeParametersInGroupAsync(parameterGroup.DBClusterParameterGroupName,
                new List<string> { "auto_increment_offset",
"auto_increment_increment" });

            await ModifyParametersAsync(parameterGroup.DBClusterParameterGroupName,
parameters);

            await
DescribeUserSourceParameters(parameterGroup.DBClusterParameterGroupName);

            var engineVersionChoice = await
ChooseDBEngineVersionAsync(parameterGroupFamily);

            var newClusterIdentifier = "Example-Cluster-" + DateTime.Now.Ticks;

            newCluster = await CreateNewCluster
            (
                parameterGroup,
                engine,
                engineVersionChoice.EngineVersion,
                newClusterIdentifier
            );

            var instanceClassChoice = await ChooseDBInstanceClass(engine,
engineVersionChoice.EngineVersion);

            var newInstanceIdentifier = "Example-Instance-" + DateTime.Now.Ticks;

            newInstance = await CreateNewInstance(
                newClusterIdentifier,
                engine,
                engineVersionChoice.EngineVersion,
                instanceClassChoice.DBInstanceClass,
                newInstanceIdentifier
            );

            DisplayConnectionString(newCluster!);
            await CreateSnapshot(newCluster!);
            await CleanupResources(newInstance, newCluster, parameterGroup);

            Console.WriteLine("Scenario complete.");
            Console.WriteLine(sepBar);
```

```
        }
        catch (Exception ex)
        {
            await CleanupResources(newInstance, newCluster, parameterGroup);
            logger.LogError(ex, "There was a problem executing the scenario.");
        }
    }

    /// <summary>
    /// Choose the Aurora DB parameter group family from a list of available
options.
    /// </summary>
    /// <returns>The selected parameter group family.</returns>
    public static async Task<string> ChooseParameterGroupFamilyAsync()
    {
        Console.WriteLine(sepBar);
        // 1. Get a list of available engines.
        var engines = await
auroraWrapper.DescribeDBEngineVersionsForEngineAsync(engine);

        Console.WriteLine($"1. The following is a list of available DB parameter
group families for engine {engine}:");

        var parameterGroupFamilies =
            engines.GroupBy(e => e.DBParameterGroupFamily).ToList();
        for (var i = 1; i <= parameterGroupFamilies.Count; i++)
        {
            var parameterGroupFamily = parameterGroupFamilies[i - 1];
            // List the available parameter group families.
            Console.WriteLine(
                $"\t{i}. Family: {parameterGroupFamily.Key}");
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > parameterGroupFamilies.Count)
        {
            Console.WriteLine("2. Select an available DB parameter group family by
entering a number from the preceding list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }
        var parameterGroupFamilyChoice = parameterGroupFamilies[choiceNumber - 1];
        Console.WriteLine(sepBar);
        return parameterGroupFamilyChoice.Key;
```

```
    }

    /// <summary>
    /// Create and get information on a DB parameter group.
    /// </summary>
    /// <param name="dbParameterGroupFamily">The DBParameterGroupFamily for the new
 DB parameter group.</param>
    /// <returns>The new DBParameterGroup.</returns>
    public static async Task<DBClusterParameterGroup>
 CreateDBParameterGroupAsync(string dbParameterGroupFamily)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine($"2. Create new DB parameter group with family
 {dbParameterGroupFamily}:");

        var parameterGroup = await
 auroraWrapper.CreateCustomClusterParameterGroupAsync(
            dbParameterGroupFamily,
            "ExampleParameterGroup-" + DateTime.Now.Ticks,
            "New example parameter group");

        var groupInfo =
            await
 auroraWrapper.DescribeCustomDBClusterParameterGroupAsync(parameterGroup.DBClusterParameterG

        Console.WriteLine(
            $"3. New DB parameter group created: \n\t{groupInfo?.Description}, \n
\tARN {groupInfo?.DBClusterParameterGroupName}");
        Console.WriteLine(sepBar);
        return parameterGroup;
    }

    /// <summary>
    /// Get and describe parameters from a DBParameterGroup.
    /// </summary>
    /// <param name="parameterGroupName">The name of the DBParameterGroup.</param>
    /// <param name="parameterNames">Optional specific names of parameters to
 describe.</param>
    /// <returns>The list of requested parameters.</returns>
    public static async Task<List<Parameter>> DescribeParametersInGroupAsync(string
 parameterGroupName, List<string>? parameterNames = null)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine("4. Get some parameters from the group.");
```

```
        Console.WriteLine(sepBar);

        var parameters =
            await
auroraWrapper.DescribeDBClusterParametersInGroupAsync(parameterGroupName);

        var matchingParameters =
            parameters.Where(p => parameterNames == null ||
parameterNames.Contains(p.ParameterName)).ToList();

        Console.WriteLine("5. Parameter information:");
        matchingParameters.ForEach(p =>
            Console.WriteLine(
                $"\n\tParameter: {p.ParameterName}." +
                $"\n\tDescription: {p.Description}." +
                $"\n\tAllowed Values: {p.AllowedValues}." +
                $"\n\tValue: {p.ParameterValue}."));

        Console.WriteLine(sepBar);

        return matchingParameters;
    }

    /// <summary>
    /// Modify a parameter from a DBParameterGroup.
    /// </summary>
    /// <param name="parameterGroupName">Name of the DBParameterGroup.</param>
    /// <param name="parameters">The parameters to modify.</param>
    /// <returns>Async task.</returns>
    public static async Task ModifyParametersAsync(string parameterGroupName,
List<Parameter> parameters)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine("6. Modify some parameters in the group.");

        await auroraWrapper.ModifyIntegerParametersInGroupAsync(parameterGroupName,
parameters);

        Console.WriteLine(sepBar);
    }

    /// <summary>
    /// Describe the user source parameters in the group.
    /// </summary>
```

```
    /// <param name="parameterGroupName">The name of the DBParameterGroup.</param>
    /// <returns>Async task.</returns>
    public static async Task DescribeUserSourceParameters(string parameterGroupName)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine("7. Describe updated user source parameters in the
group.");

        var parameters =
            await
auroraWrapper.DescribeDBClusterParametersInGroupAsync(parameterGroupName, "user");

        parameters.ForEach(p =>
            Console.WriteLine(
                $"\n\tParameter: {p.ParameterName}." +
                $"\n\tDescription: {p.Description}." +
                $"\n\tAllowed Values: {p.AllowedValues}." +
                $"\n\tValue: {p.ParameterValue}."));

        Console.WriteLine(sepBar);
    }

    /// <summary>
    /// Choose a DB engine version.
    /// </summary>
    /// <param name="dbParameterGroupFamily">DB parameter group family for engine
choice.</param>
    /// <returns>The selected engine version.</returns>
    public static async Task<DBEngineVersion> ChooseDBEngineVersionAsync(string
dbParameterGroupFamily)
    {
        Console.WriteLine(sepBar);
        // Get a list of allowed engines.
        var allowedEngines =
            await auroraWrapper.DescribeDBEngineVersionsForEngineAsync(engine,
dbParameterGroupFamily);

        Console.WriteLine($"Available DB engine versions for parameter group family
{dbParameterGroupFamily}:");
        int i = 1;
        foreach (var version in allowedEngines)
        {
            Console.WriteLine(
                $"\t{i}. {version.DBEngineVersionDescription}.");
```

```
            i++;
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > allowedEngines.Count)
        {
            Console.WriteLine("8. Select an available DB engine version by entering
 a number from the list above:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }

        var engineChoice = allowedEngines[choiceNumber - 1];
        Console.WriteLine(sepBar);
        return engineChoice;
    }

    /// <summary>
    /// Create a new RDS DB cluster.
    /// </summary>
    /// <param name="parameterGroup">Parameter group to use for the DB cluster.</
param>
    /// <param name="engineName">Engine to use for the DB cluster.</param>
    /// <param name="engineVersion">Engine version to use for the DB cluster.</
param>
    /// <param name="clusterIdentifier">Cluster identifier to use for the DB
 cluster.</param>
    /// <returns>The new DB cluster.</returns>
    public static async Task<DBCluster?> CreateNewCluster(DBClusterParameterGroup
 parameterGroup,
        string engineName, string engineVersion, string clusterIdentifier)
    {
        Console.WriteLine(sepBar);
        Console.WriteLine($"9. Create a new DB cluster with identifier
 {clusterIdentifier}.");

        DBCluster newCluster;
        var clusters = await auroraWrapper.DescribeDBClustersPagedAsync();
        var isClusterCreated = clusters.Any(i => i.DBClusterIdentifier ==
 clusterIdentifier);

        if (isClusterCreated)
        {
            Console.WriteLine("Cluster already created.");
```

```
            newCluster = clusters.First(i => i.DBClusterIdentifier ==
 clusterIdentifier);
        }
        else
        {
            Console.WriteLine("Enter an admin username:");
            var username = Console.ReadLine();

            Console.WriteLine("Enter an admin password:");
            var password = Console.ReadLine();

            newCluster = await auroraWrapper.CreateDBClusterWithAdminAsync(
                "ExampleDatabase",
                clusterIdentifier,
                parameterGroup.DBClusterParameterGroupName,
                engineName,
                engineVersion,
                username!,
                password!
            );

            Console.WriteLine("10. Waiting for DB cluster to be ready...");
            while (newCluster.Status != "available")
            {
                Console.Write(".");
                Thread.Sleep(5000);
                clusters = await
 auroraWrapper.DescribeDBClustersPagedAsync(clusterIdentifier);
                newCluster = clusters.First();
            }
        }

        Console.WriteLine(sepBar);
        return newCluster;
    }

    /// <summary>
    /// Choose a DB instance class for a particular engine and engine version.
    /// </summary>
    /// <param name="engine">DB engine for DB instance choice.</param>
    /// <param name="engineVersion">DB engine version for DB instance choice.</
param>
    /// <returns>The selected orderable DB instance option.</returns>
```

```
    public static async Task<OrderableDBInstanceOption> ChooseDBInstanceClass(string
 engine, string engineVersion)
    {
        Console.WriteLine(sepBar);
        // Get a list of allowed DB instance classes.
        var allowedInstances =
            await auroraWrapper.DescribeOrderableDBInstanceOptionsPagedAsync(engine,
 engineVersion);

        Console.WriteLine($"Available DB instance classes for engine {engine} and
 version {engineVersion}:");
        int i = 1;

        foreach (var instance in allowedInstances)
        {
            Console.WriteLine(
                $"\t{i}. Instance class: {instance.DBInstanceClass} (storage type
 {instance.StorageType})");
            i++;
        }

        var choiceNumber = 0;
        while (choiceNumber < 1 || choiceNumber > allowedInstances.Count)
        {
            Console.WriteLine("11. Select an available DB instance class by entering
 a number from the preceding list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out choiceNumber);
        }

        var instanceChoice = allowedInstances[choiceNumber - 1];
        Console.WriteLine(sepBar);
        return instanceChoice;
    }

    /// <summary>
    /// Create a new DB instance.
    /// </summary>
    /// <param name="engineName">Engine to use for the DB instance.</param>
    /// <param name="engineVersion">Engine version to use for the DB instance.</
param>
    /// <param name="instanceClass">Instance class to use for the DB instance.</
param>
```

```
   /// <param name="instanceIdentifier">Instance identifier to use for the DB
instance.</param>
   /// <returns>The new DB instance.</returns>
   public static async Task<DBInstance?> CreateNewInstance(
       string clusterIdentifier,
       string engineName,
       string engineVersion,
       string instanceClass,
       string instanceIdentifier)
   {
       Console.WriteLine(sepBar);
       Console.WriteLine($"12. Create a new DB instance with identifier
{instanceIdentifier}.");
       bool isInstanceReady = false;
       DBInstance newInstance;
       var instances = await auroraWrapper.DescribeDBInstancesPagedAsync();
       isInstanceReady = instances.FirstOrDefault(i =>
           i.DBInstanceIdentifier == instanceIdentifier)?.DBInstanceStatus ==
"available";

       if (isInstanceReady)
       {
           Console.WriteLine("Instance already created.");
           newInstance = instances.First(i => i.DBInstanceIdentifier ==
instanceIdentifier);
       }
       else
       {
           newInstance = await auroraWrapper.CreateDBInstanceInClusterAsync(
               clusterIdentifier,
               instanceIdentifier,
               engineName,
               engineVersion,
               instanceClass
           );

           Console.WriteLine("13. Waiting for DB instance to be ready...");
           while (!isInstanceReady)
           {
               Console.Write(".");
               Thread.Sleep(5000);
               instances = await
auroraWrapper.DescribeDBInstancesPagedAsync(instanceIdentifier);
```

```
                isInstanceReady = instances.FirstOrDefault()?.DBInstanceStatus ==
 "available";
                newInstance = instances.First();
            }
        }

        Console.WriteLine(sepBar);
        return newInstance;
    }

    /// <summary>
    /// Display a connection string for an Amazon RDS DB cluster.
    /// </summary>
    /// <param name="cluster">The DB cluster to use to get a connection string.</
param>
    public static void DisplayConnectionString(DBCluster cluster)
    {
        Console.WriteLine(sepBar);
        // Display the connection string.
        Console.WriteLine("14. New DB cluster connection string: ");
        Console.WriteLine(
            $"\n{engine} -h {cluster.Endpoint} -P {cluster.Port} "
            + $"-u {cluster.MasterUsername} -p [YOUR PASSWORD]\n");

        Console.WriteLine(sepBar);
    }

    /// <summary>
    /// Create a snapshot from an Amazon RDS DB cluster.
    /// </summary>
    /// <param name="cluster">DB cluster to use when creating a snapshot.</param>
    /// <returns>The snapshot object.</returns>
    public static async Task<DBClusterSnapshot> CreateSnapshot(DBCluster cluster)
    {
        Console.WriteLine(sepBar);
        // Create a snapshot.
        Console.WriteLine($"15. Creating snapshot from DB cluster
 {cluster.DBClusterIdentifier}.");
        var snapshot = await auroraWrapper.CreateClusterSnapshotByIdentifierAsync(
            cluster.DBClusterIdentifier,
            "ExampleSnapshot-" + DateTime.Now.Ticks);

        // Wait for the snapshot to be available.
        bool isSnapshotReady = false;
```

```
        Console.WriteLine($"16. Waiting for snapshot to be ready...");
        while (!isSnapshotReady)
        {
            Console.Write(".");
            Thread.Sleep(5000);
            var snapshots =
                await
auroraWrapper.DescribeDBClusterSnapshotsByIdentifierAsync(cluster.DBClusterIdentifier);
            isSnapshotReady = snapshots.FirstOrDefault()?.Status == "available";
            snapshot = snapshots.First();
        }

        Console.WriteLine(
            $"Snapshot {snapshot.DBClusterSnapshotIdentifier} status is
{snapshot.Status}.");
        Console.WriteLine(sepBar);
        return snapshot;
    }

    /// <summary>
    /// Clean up resources from the scenario.
    /// </summary>
    /// <param name="newInstance">The instance to clean up.</param>
    /// <param name="newCluster">The cluster to clean up.</param>
    /// <param name="parameterGroup">The parameter group to clean up.</param>
    /// <returns>Async Task.</returns>
    private static async Task CleanupResources(
        DBInstance? newInstance,
        DBCluster? newCluster,
        DBClusterParameterGroup? parameterGroup)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"Clean up resources.");

        if (newInstance is not null && GetYesNoResponse($"\tClean up instance
{newInstance.DBInstanceIdentifier}? (y/n)"))
        {
            // Delete the DB instance.
            Console.WriteLine($"17. Deleting the DB instance
{newInstance.DBInstanceIdentifier}.");
            await
auroraWrapper.DeleteDBInstanceByIdentifierAsync(newInstance.DBInstanceIdentifier);
        }
```

```
        if (newCluster is not null && GetYesNoResponse($"\tClean up cluster
{newCluster.DBClusterIdentifier}? (y/n)"))
        {
            // Delete the DB cluster.
            Console.WriteLine($"18. Deleting the DB cluster
{newCluster.DBClusterIdentifier}.");
            await
auroraWrapper.DeleteDBClusterByIdentifierAsync(newCluster.DBClusterIdentifier);

            // Wait for the DB cluster to delete.
            Console.WriteLine($"19. Waiting for the DB cluster to delete...");
            bool isClusterDeleted = false;

            while (!isClusterDeleted)
            {
                Console.Write(".");
                Thread.Sleep(5000);
                var cluster = await auroraWrapper.DescribeDBClustersPagedAsync();
                isClusterDeleted = cluster.All(i => i.DBClusterIdentifier !=
newCluster.DBClusterIdentifier);
            }

            Console.WriteLine("DB cluster deleted.");
        }

        if (parameterGroup is not null && GetYesNoResponse($"\tClean up parameter
group? (y/n)"))
        {
            Console.WriteLine($"20. Deleting the DB parameter group
{parameterGroup.DBClusterParameterGroupName}.");
            await
auroraWrapper.DeleteClusterParameterGroupByNameAsync(parameterGroup.DBClusterParameterGroup
            Console.WriteLine("Parameter group deleted.");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Get a yes or no response from the user.
    /// </summary>
    /// <param name="question">The question string to print on the console.</param>
    /// <returns>True if the user responds with a yes.</returns>
```

```
    private static bool GetYesNoResponse(string question)
    {
        Console.WriteLine(question);
        var ynResponse = Console.ReadLine();
        var response = ynResponse != null &&
                       ynResponse.Equals("y",
                           StringComparison.InvariantCultureIgnoreCase);
        return response;
    }
```

Métodos de encapsulamento que são chamados pelo cenário para gerenciar as ações do Aurora.

```
using Amazon.RDS;
using Amazon.RDS.Model;

namespace AuroraActions;

/// <summary>
/// Wrapper for the Amazon Aurora cluster client operations.
/// </summary>
public class AuroraWrapper
{
    private readonly IAmazonRDS _amazonRDS;
    public AuroraWrapper(IAmazonRDS amazonRDS)
    {
        _amazonRDS = amazonRDS;
    }

    /// <summary>
    /// Get a list of DB engine versions for a particular DB engine.
    /// </summary>
    /// <param name="engine">The name of the engine.</param>
    /// <param name="parameterGroupFamily">Optional parameter group family name.</
param>
    /// <returns>A list of DBEngineVersions.</returns>
    public async Task<List<DBEngineVersion>>
 DescribeDBEngineVersionsForEngineAsync(string engine,
        string? parameterGroupFamily = null)
    {
        var response = await _amazonRDS.DescribeDBEngineVersionsAsync(
            new DescribeDBEngineVersionsRequest()
```

```
            {
                Engine = engine,
                DBParameterGroupFamily = parameterGroupFamily
            });
        return response.DBEngineVersions;
    }

    /// <summary>
    /// Create a custom cluster parameter group.
    /// </summary>
    /// <param name="parameterGroupFamily">The family of the parameter group.</
param>
    /// <param name="groupName">The name for the new parameter group.</param>
    /// <param name="description">A description for the new parameter group.</param>
    /// <returns>The new parameter group object.</returns>
    public async Task<DBClusterParameterGroup>
 CreateCustomClusterParameterGroupAsync(
        string parameterGroupFamily,
        string groupName,
        string description)
    {
        var request = new CreateDBClusterParameterGroupRequest
        {
            DBParameterGroupFamily = parameterGroupFamily,
            DBClusterParameterGroupName = groupName,
            Description = description,
        };

        var response = await _amazonRDS.CreateDBClusterParameterGroupAsync(request);
        return response.DBClusterParameterGroup;
    }

    /// <summary>
    /// Describe the cluster parameters in a parameter group.
    /// </summary>
    /// <param name="groupName">The name of the parameter group.</param>
    /// <param name="source">The optional name of the source filter.</param>
    /// <returns>The collection of parameters.</returns>
    public async Task<List<Parameter>>
 DescribeDBClusterParametersInGroupAsync(string groupName, string? source = null)
    {
        var paramList = new List<Parameter>();

        DescribeDBClusterParametersResponse response;
```

```
        var request = new DescribeDBClusterParametersRequest
        {
            DBClusterParameterGroupName = groupName,
            Source = source,
        };

        // Get the full list if there are multiple pages.
        do
        {
            response = await _amazonRDS.DescribeDBClusterParametersAsync(request);
            paramList.AddRange(response.Parameters);

            request.Marker = response.Marker;
        }
        while (response.Marker is not null);

        return paramList;
    }

    /// <summary>
    /// Get the description of a DB cluster parameter group by name.
    /// </summary>
    /// <param name="name">The name of the DB parameter group to describe.</param>
    /// <returns>The parameter group description.</returns>
    public async Task<DBClusterParameterGroup?>
DescribeCustomDBClusterParameterGroupAsync(string name)
    {
        var response = await _amazonRDS.DescribeDBClusterParameterGroupsAsync(
            new DescribeDBClusterParameterGroupsRequest()
            {
                DBClusterParameterGroupName = name
            });
        return response.DBClusterParameterGroups.FirstOrDefault();
    }

    /// <summary>
    /// Modify the specified integer parameters with new values from user input.
    /// </summary>
    /// <param name="groupName">The group name for the parameters.</param>
    /// <param name="parameters">The list of integer parameters to modify.</param>
    /// <param name="newValue">Optional int value to set for parameters.</param>
    /// <returns>The name of the group that was modified.</returns>
    public async Task<string> ModifyIntegerParametersInGroupAsync(string groupName,
List<Parameter> parameters, int newValue = 0)
```

```
    {
        foreach (var p in parameters)
        {
            if (p.IsModifiable.GetValueOrDefault() && p.DataType == "integer")
            {
                while (newValue == 0)
                {
                    Console.WriteLine(
                        $"Enter a new value for {p.ParameterName} from the allowed
values {p.AllowedValues} ");

                    var choice = Console.ReadLine();
                    int.TryParse(choice, out newValue);
                }

                p.ParameterValue = newValue.ToString();
            }
        }

        var request = new ModifyDBClusterParameterGroupRequest
        {
            Parameters = parameters,
            DBClusterParameterGroupName = groupName,
        };

        var result = await _amazonRDS.ModifyDBClusterParameterGroupAsync(request);
        return result.DBClusterParameterGroupName;
    }


    /// <summary>
    /// Get a list of orderable DB instance options for a specific
    /// engine and engine version.
    /// </summary>
    /// <param name="engine">Name of the engine.</param>
    /// <param name="engineVersion">Version of the engine.</param>
    /// <returns>List of OrderableDBInstanceOptions.</returns>
    public async Task<List<OrderableDBInstanceOption>>
DescribeOrderableDBInstanceOptionsPagedAsync(string engine, string engineVersion)
    {
        // Use a paginator to get a list of DB instance options.
        var results = new List<OrderableDBInstanceOption>();
        var paginateInstanceOptions =
_amazonRDS.Paginators.DescribeOrderableDBInstanceOptions(
```

```
            new DescribeOrderableDBInstanceOptionsRequest()
            {
                Engine = engine,
                EngineVersion = engineVersion,
            });
        // Get the entire list using the paginator.
        await foreach (var instanceOptions in
paginateInstanceOptions.OrderableDBInstanceOptions)
        {
            results.Add(instanceOptions);
        }
        return results;
    }

    /// <summary>
    /// Delete a particular parameter group by name.
    /// </summary>
    /// <param name="groupName">The name of the parameter group.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteClusterParameterGroupByNameAsync(string groupName)
    {
        var request = new DeleteDBClusterParameterGroupRequest
        {
            DBClusterParameterGroupName = groupName,
        };

        var response = await _amazonRDS.DeleteDBClusterParameterGroupAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }

    /// <summary>
    /// Create a new cluster and database.
    /// </summary>
    /// <param name="dbName">The name of the new database.</param>
    /// <param name="clusterIdentifier">The identifier of the cluster.</param>
    /// <param name="parameterGroupName">The name of the parameter group.</param>
    /// <param name="dbEngine">The engine to use for the new cluster.</param>
    /// <param name="dbEngineVersion">The version of the engine to use.</param>
    /// <param name="adminName">The admin username.</param>
    /// <param name="adminPassword">The primary admin password.</param>
    /// <returns>The cluster object.</returns>
    public async Task<DBCluster> CreateDBClusterWithAdminAsync(
        string dbName,
        string clusterIdentifier,
```

```
        string parameterGroupName,
        string dbEngine,
        string dbEngineVersion,
        string adminName,
        string adminPassword)
    {
        var request = new CreateDBClusterRequest
        {
            DatabaseName = dbName,
            DBClusterIdentifier = clusterIdentifier,
            DBClusterParameterGroupName = parameterGroupName,
            Engine = dbEngine,
            EngineVersion = dbEngineVersion,
            MasterUsername = adminName,
            MasterUserPassword = adminPassword,
        };

        var response = await _amazonRDS.CreateDBClusterAsync(request);
        return response.DBCluster;
    }

    /// <summary>
    /// Returns a list of DB instances.
    /// </summary>
    /// <param name="dbInstanceIdentifier">Optional name of a specific DB
instance.</param>
    /// <returns>List of DB instances.</returns>
    public async Task<List<DBInstance>> DescribeDBInstancesPagedAsync(string?
dbInstanceIdentifier = null)
    {
        var results = new List<DBInstance>();
        var instancesPaginator = _amazonRDS.Paginators.DescribeDBInstances(
            new DescribeDBInstancesRequest
            {
                DBInstanceIdentifier = dbInstanceIdentifier
            });
        // Get the entire list using the paginator.
        await foreach (var instances in instancesPaginator.DBInstances)
        {
            results.Add(instances);
        }
        return results;
    }
```

```
    /// <summary>
    /// Returns a list of DB clusters.
    /// </summary>
    /// <param name="dbInstanceIdentifier">Optional name of a specific DB cluster.</
param>
    /// <returns>List of DB clusters.</returns>
    public async Task<List<DBCluster>> DescribeDBClustersPagedAsync(string?
 dbClusterIdentifier = null)
    {
        var results = new List<DBCluster>();

        DescribeDBClustersResponse response;
        DescribeDBClustersRequest request = new DescribeDBClustersRequest
        {
            DBClusterIdentifier = dbClusterIdentifier
        };
        // Get the full list if there are multiple pages.
        do
        {
            response = await _amazonRDS.DescribeDBClustersAsync(request);
            if (response.DBClusters != null)
            {
                results.AddRange(response.DBClusters);
            }
            request.Marker = response.Marker;
        }
        while (response.Marker is not null);
        return results;
    }

    /// <summary>
    /// Create an Amazon Relational Database Service (Amazon RDS) DB instance
    /// with a particular set of properties. Use the action DescribeDBInstancesAsync
    /// to determine when the DB instance is ready to use.
    /// </summary>
    /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <param name="dbEngine">The engine for the DB instance.</param>
    /// <param name="dbEngineVersion">Version for the DB instance.</param>
    /// <param name="instanceClass">Class for the DB instance.</param>
    /// <returns>DB instance object.</returns>
    public async Task<DBInstance> CreateDBInstanceInClusterAsync(
        string dbClusterIdentifier,
        string dbInstanceIdentifier,
```

```
        string dbEngine,
        string dbEngineVersion,
        string instanceClass)
    {
        // When creating the instance within a cluster, do not specify the name or
size.
        var response = await _amazonRDS.CreateDBInstanceAsync(
            new CreateDBInstanceRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                DBInstanceIdentifier = dbInstanceIdentifier,
                Engine = dbEngine,
                EngineVersion = dbEngineVersion,
                DBInstanceClass = instanceClass
            });

        return response.DBInstance;
    }

    /// <summary>
    /// Create a snapshot of a cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <param name="snapshotIdentifier">Identifier for the snapshot.</param>
    /// <returns>DB snapshot object.</returns>
    public async Task<DBClusterSnapshot>
CreateClusterSnapshotByIdentifierAsync(string dbClusterIdentifier, string
snapshotIdentifier)
    {
        var response = await _amazonRDS.CreateDBClusterSnapshotAsync(
            new CreateDBClusterSnapshotRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                DBClusterSnapshotIdentifier = snapshotIdentifier,
            });

        return response.DBClusterSnapshot;
    }

    /// <summary>
    /// Return a list of DB snapshots for a particular DB cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <returns>List of DB snapshots.</returns>
```

```
    public async Task<List<DBClusterSnapshot>>
DescribeDBClusterSnapshotsByIdentifierAsync(string dbClusterIdentifier)
    {
        var results = new List<DBClusterSnapshot>();

        DescribeDBClusterSnapshotsResponse response;
        DescribeDBClusterSnapshotsRequest request = new
DescribeDBClusterSnapshotsRequest
        {
            DBClusterIdentifier = dbClusterIdentifier
        };
        // Get the full list if there are multiple pages.
        do
        {
            response = await _amazonRDS.DescribeDBClusterSnapshotsAsync(request);
            results.AddRange(response.DBClusterSnapshots);
            request.Marker = response.Marker;
        }
        while (response.Marker is not null);
        return results;
    }

    /// <summary>
    /// Delete a particular DB cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <returns>DB cluster object.</returns>
    public async Task<DBCluster> DeleteDBClusterByIdentifierAsync(string
dbClusterIdentifier)
    {
        var response = await _amazonRDS.DeleteDBClusterAsync(
            new DeleteDBClusterRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                SkipFinalSnapshot = true
            });

        return response.DBCluster;
    }

    /// <summary>
    /// Delete a particular DB instance.
    /// </summary>
    /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
```

```
    /// <returns>DB instance object.</returns>
    public async Task<DBInstance> DeleteDBInstanceByIdentifierAsync(string
 dbInstanceIdentifier)
    {
        var response = await _amazonRDS.DeleteDBInstanceAsync(
            new DeleteDBInstanceRequest()
            {
                DBInstanceIdentifier = dbInstanceIdentifier,
                SkipFinalSnapshot = true,
                DeleteAutomatedBackups = true
            });

        return response.DBInstance;
    }
}
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para .NET .
  - CriarDBCluster
  - CriarDBClusterParameterGroup
  - Criar DBCluster instantâneo
  - CriarDBInstance
  - ExcluirDBCluster
  - ExcluirDBClusterParameterGroup
  - ExcluirDBInstance
  - DescreverDBClusterParameterGroups
  - Descreva DBCluster os parâmetros
  - Descreva os DBCluster instantâneos
  - DescreverDBClusters
  - Descreva DBEngine as versões
  - DescreverDBInstances
  - DescribeOrderableDBInstanceOpções
  - ModifiqueDBClusterParameterGroup

Ações

`CreateDBCluster`

O código de exemplo a seguir mostra como usar `CreateDBCluster`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
/// <summary>
/// Create a new cluster and database.
/// </summary>
/// <param name="dbName">The name of the new database.</param>
/// <param name="clusterIdentifier">The identifier of the cluster.</param>
/// <param name="parameterGroupName">The name of the parameter group.</param>
/// <param name="dbEngine">The engine to use for the new cluster.</param>
/// <param name="dbEngineVersion">The version of the engine to use.</param>
/// <param name="adminName">The admin username.</param>
/// <param name="adminPassword">The primary admin password.</param>
/// <returns>The cluster object.</returns>
public async Task<DBCluster> CreateDBClusterWithAdminAsync(
    string dbName,
    string clusterIdentifier,
    string parameterGroupName,
    string dbEngine,
    string dbEngineVersion,
    string adminName,
    string adminPassword)
{
    var request = new CreateDBClusterRequest
    {
        DatabaseName = dbName,
        DBClusterIdentifier = clusterIdentifier,
        DBClusterParameterGroupName = parameterGroupName,
        Engine = dbEngine,
        EngineVersion = dbEngineVersion,
        MasterUsername = adminName,
```

```
            MasterUserPassword = adminPassword,
        };

        var response = await _amazonRDS.CreateDBClusterAsync(request);
        return response.DBCluster;
    }
```

- Para obter detalhes da API, consulte Criar DBCluster na referência AWS SDK para .NET da API.

`CreateDBClusterParameterGroup`

O código de exemplo a seguir mostra como usar `CreateDBClusterParameterGroup`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Create a custom cluster parameter group.
    /// </summary>
    /// <param name="parameterGroupFamily">The family of the parameter group.</
param>
    /// <param name="groupName">The name for the new parameter group.</param>
    /// <param name="description">A description for the new parameter group.</param>
    /// <returns>The new parameter group object.</returns>
    public async Task<DBClusterParameterGroup>
 CreateCustomClusterParameterGroupAsync(
        string parameterGroupFamily,
        string groupName,
        string description)
    {
        var request = new CreateDBClusterParameterGroupRequest
        {
            DBParameterGroupFamily = parameterGroupFamily,
            DBClusterParameterGroupName = groupName,
```

```
            Description = description,
        };

        var response = await _amazonRDS.CreateDBClusterParameterGroupAsync(request);
        return response.DBClusterParameterGroup;
    }
```

- Para obter detalhes da API, consulte Criar DBCluster ParameterGroup na referência AWS SDK para .NET da API.

`CreateDBClusterSnapshot`

O código de exemplo a seguir mostra como usar `CreateDBClusterSnapshot`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Create a snapshot of a cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <param name="snapshotIdentifier">Identifier for the snapshot.</param>
    /// <returns>DB snapshot object.</returns>
    public async Task<DBClusterSnapshot>
CreateClusterSnapshotByIdentifierAsync(string dbClusterIdentifier, string
snapshotIdentifier)
    {
        var response = await _amazonRDS.CreateDBClusterSnapshotAsync(
            new CreateDBClusterSnapshotRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                DBClusterSnapshotIdentifier = snapshotIdentifier,
            });

        return response.DBClusterSnapshot;
```

```
    }
```

- Para obter detalhes da API, consulte Criar DBCluster instantâneo na Referência AWS SDK para .NET da API.

`CreateDBInstance`

O código de exemplo a seguir mostra como usar `CreateDBInstance`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Create an Amazon Relational Database Service (Amazon RDS) DB instance
    /// with a particular set of properties. Use the action DescribeDBInstancesAsync
    /// to determine when the DB instance is ready to use.
    /// </summary>
    /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <param name="dbEngine">The engine for the DB instance.</param>
    /// <param name="dbEngineVersion">Version for the DB instance.</param>
    /// <param name="instanceClass">Class for the DB instance.</param>
    /// <returns>DB instance object.</returns>
    public async Task<DBInstance> CreateDBInstanceInClusterAsync(
        string dbClusterIdentifier,
        string dbInstanceIdentifier,
        string dbEngine,
        string dbEngineVersion,
        string instanceClass)
    {
        // When creating the instance within a cluster, do not specify the name or
size.
        var response = await _amazonRDS.CreateDBInstanceAsync(
            new CreateDBInstanceRequest()
            {
```

```
                DBClusterIdentifier = dbClusterIdentifier,
                DBInstanceIdentifier = dbInstanceIdentifier,
                Engine = dbEngine,
                EngineVersion = dbEngineVersion,
                DBInstanceClass = instanceClass
            });

        return response.DBInstance;
    }
```

- Para obter detalhes da API, consulte Criar DBInstance na referência AWS SDK para .NET da API.

`DeleteDBCluster`

O código de exemplo a seguir mostra como usar `DeleteDBCluster`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Delete a particular DB cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <returns>DB cluster object.</returns>
    public async Task<DBCluster> DeleteDBClusterByIdentifierAsync(string
dbClusterIdentifier)
    {
        var response = await _amazonRDS.DeleteDBClusterAsync(
            new DeleteDBClusterRequest()
            {
                DBClusterIdentifier = dbClusterIdentifier,
                SkipFinalSnapshot = true
            });
```

```
        return response.DBCluster;
    }
```

- Para obter detalhes da API, consulte Excluir DBCluster na Referência AWS SDK para .NET da API.

`DeleteDBClusterParameterGroup`

O código de exemplo a seguir mostra como usar `DeleteDBClusterParameterGroup`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Delete a particular parameter group by name.
    /// </summary>
    /// <param name="groupName">The name of the parameter group.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteClusterParameterGroupByNameAsync(string groupName)
    {
        var request = new DeleteDBClusterParameterGroupRequest
        {
            DBClusterParameterGroupName = groupName,
        };

        var response = await _amazonRDS.DeleteDBClusterParameterGroupAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- Para obter detalhes da API, consulte Excluir DBCluster ParameterGroup na Referência AWS SDK para .NET da API.

`DeleteDBInstance`

O código de exemplo a seguir mostra como usar `DeleteDBInstance`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Delete a particular DB instance.
    /// </summary>
    /// <param name="dbInstanceIdentifier">DB instance identifier.</param>
    /// <returns>DB instance object.</returns>
    public async Task<DBInstance> DeleteDBInstanceByIdentifierAsync(string
dbInstanceIdentifier)
    {
        var response = await _amazonRDS.DeleteDBInstanceAsync(
            new DeleteDBInstanceRequest()
            {
                DBInstanceIdentifier = dbInstanceIdentifier,
                SkipFinalSnapshot = true,
                DeleteAutomatedBackups = true
            });

        return response.DBInstance;
    }
```

- Para obter detalhes da API, consulte Excluir DBInstance na Referência AWS SDK para .NET da API.

`DescribeDBClusterParameterGroups`

O código de exemplo a seguir mostra como usar `DescribeDBClusterParameterGroups`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Get the description of a DB cluster parameter group by name.
    /// </summary>
    /// <param name="name">The name of the DB parameter group to describe.</param>
    /// <returns>The parameter group description.</returns>
    public async Task<DBClusterParameterGroup?>
DescribeCustomDBClusterParameterGroupAsync(string name)
    {
        var response = await _amazonRDS.DescribeDBClusterParameterGroupsAsync(
            new DescribeDBClusterParameterGroupsRequest()
            {
                DBClusterParameterGroupName = name
            });
        return response.DBClusterParameterGroups.FirstOrDefault();
    }
```

- Para obter detalhes da API, consulte Descrever DBCluster ParameterGroups na Referência AWS SDK para .NET da API.

`DescribeDBClusterParameters`

O código de exemplo a seguir mostra como usar `DescribeDBClusterParameters`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Describe the cluster parameters in a parameter group.
    /// </summary>
    /// <param name="groupName">The name of the parameter group.</param>
    /// <param name="source">The optional name of the source filter.</param>
    /// <returns>The collection of parameters.</returns>
    public async Task<List<Parameter>>
DescribeDBClusterParametersInGroupAsync(string groupName, string? source = null)
    {
        var paramList = new List<Parameter>();

        DescribeDBClusterParametersResponse response;
        var request = new DescribeDBClusterParametersRequest
        {
            DBClusterParameterGroupName = groupName,
            Source = source,
        };

        // Get the full list if there are multiple pages.
        do
        {
            response = await _amazonRDS.DescribeDBClusterParametersAsync(request);
            paramList.AddRange(response.Parameters);

            request.Marker = response.Marker;
        }
        while (response.Marker is not null);

        return paramList;
    }
```

- Para obter detalhes da API, consulte Descrever DBCluster os parâmetros na Referência AWS SDK para .NET da API.

`DescribeDBClusterSnapshots`

O código de exemplo a seguir mostra como usar `DescribeDBClusterSnapshots`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Return a list of DB snapshots for a particular DB cluster.
    /// </summary>
    /// <param name="dbClusterIdentifier">DB cluster identifier.</param>
    /// <returns>List of DB snapshots.</returns>
    public async Task<List<DBClusterSnapshot>>
DescribeDBClusterSnapshotsByIdentifierAsync(string dbClusterIdentifier)
    {
        var results = new List<DBClusterSnapshot>();

        DescribeDBClusterSnapshotsResponse response;
        DescribeDBClusterSnapshotsRequest request = new
DescribeDBClusterSnapshotsRequest
        {
            DBClusterIdentifier = dbClusterIdentifier
        };
        // Get the full list if there are multiple pages.
        do
        {
            response = await _amazonRDS.DescribeDBClusterSnapshotsAsync(request);
            results.AddRange(response.DBClusterSnapshots);
            request.Marker = response.Marker;
        }
        while (response.Marker is not null);
        return results;
    }
```

- Para obter detalhes da API, consulte Descrever DBCluster instantâneos na Referência AWS SDK para .NET da API.

`DescribeDBClusters`

O código de exemplo a seguir mostra como usar `DescribeDBClusters`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Returns a list of DB clusters.
    /// </summary>
    /// <param name="dbInstanceIdentifier">Optional name of a specific DB cluster.</
param>
    /// <returns>List of DB clusters.</returns>
    public async Task<List<DBCluster>> DescribeDBClustersPagedAsync(string?
 dbClusterIdentifier = null)
    {
        var results = new List<DBCluster>();

        DescribeDBClustersResponse response;
        DescribeDBClustersRequest request = new DescribeDBClustersRequest
        {
            DBClusterIdentifier = dbClusterIdentifier
        };
        // Get the full list if there are multiple pages.
        do
        {
            response = await _amazonRDS.DescribeDBClustersAsync(request);
            if (response.DBClusters != null)
            {
                results.AddRange(response.DBClusters);
            }
            request.Marker = response.Marker;
        }
        while (response.Marker is not null);
        return results;
    }
```

- Para obter detalhes da API, consulte Descrever DBClusters na Referência AWS SDK para .NET da API.

`DescribeDBEngineVersions`

O código de exemplo a seguir mostra como usar `DescribeDBEngineVersions`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Get a list of DB engine versions for a particular DB engine.
    /// </summary>
    /// <param name="engine">The name of the engine.</param>
    /// <param name="parameterGroupFamily">Optional parameter group family name.</
param>
    /// <returns>A list of DBEngineVersions.</returns>
    public async Task<List<DBEngineVersion>>
 DescribeDBEngineVersionsForEngineAsync(string engine,
        string? parameterGroupFamily = null)
    {
        var response = await _amazonRDS.DescribeDBEngineVersionsAsync(
            new DescribeDBEngineVersionsRequest()
            {
                Engine = engine,
                DBParameterGroupFamily = parameterGroupFamily
            });
        return response.DBEngineVersions;
    }
```

- Para obter detalhes da API, consulte Descrever DBEngine as versões na Referência AWS SDK para .NET da API.

`DescribeDBInstances`

O código de exemplo a seguir mostra como usar `DescribeDBInstances`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
   /// <summary>
   /// Returns a list of DB instances.
   /// </summary>
   /// <param name="dbInstanceIdentifier">Optional name of a specific DB
instance.</param>
   /// <returns>List of DB instances.</returns>
   public async Task<List<DBInstance>> DescribeDBInstancesPagedAsync(string?
dbInstanceIdentifier = null)
   {
       var results = new List<DBInstance>();
       var instancesPaginator = _amazonRDS.Paginators.DescribeDBInstances(
           new DescribeDBInstancesRequest
           {
               DBInstanceIdentifier = dbInstanceIdentifier
           });
       // Get the entire list using the paginator.
       await foreach (var instances in instancesPaginator.DBInstances)
       {
           results.Add(instances);
       }
       return results;
   }
```

- Para obter detalhes da API, consulte Descrever DBInstances na Referência AWS SDK para .NET da API.

`DescribeOrderableDBInstanceOptions`

O código de exemplo a seguir mostra como usar `DescribeOrderableDBInstanceOptions`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
   /// <summary>
   /// Get a list of orderable DB instance options for a specific
   /// engine and engine version.
   /// </summary>
   /// <param name="engine">Name of the engine.</param>
   /// <param name="engineVersion">Version of the engine.</param>
   /// <returns>List of OrderableDBInstanceOptions.</returns>
   public async Task<List<OrderableDBInstanceOption>>
DescribeOrderableDBInstanceOptionsPagedAsync(string engine, string engineVersion)
   {
       // Use a paginator to get a list of DB instance options.
       var results = new List<OrderableDBInstanceOption>();
       var paginateInstanceOptions =
_amazonRDS.Paginators.DescribeOrderableDBInstanceOptions(
           new DescribeOrderableDBInstanceOptionsRequest()
           {
               Engine = engine,
               EngineVersion = engineVersion,
           });
       // Get the entire list using the paginator.
       await foreach (var instanceOptions in
paginateInstanceOptions.OrderableDBInstanceOptions)
       {
           results.Add(instanceOptions);
       }
       return results;
   }
```

- Para obter detalhes da API, consulte DescribeOrderableDBInstanceOpções na Referência AWS SDK para .NET da API.

`ModifyDBClusterParameterGroup`

O código de exemplo a seguir mostra como usar `ModifyDBClusterParameterGroup`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
   /// <summary>
   /// Modify the specified integer parameters with new values from user input.
   /// </summary>
   /// <param name="groupName">The group name for the parameters.</param>
   /// <param name="parameters">The list of integer parameters to modify.</param>
   /// <param name="newValue">Optional int value to set for parameters.</param>
   /// <returns>The name of the group that was modified.</returns>
   public async Task<string> ModifyIntegerParametersInGroupAsync(string groupName,
List<Parameter> parameters, int newValue = 0)
   {
       foreach (var p in parameters)
       {
           if (p.IsModifiable.GetValueOrDefault() && p.DataType == "integer")
           {
               while (newValue == 0)
               {
                   Console.WriteLine(
                       $"Enter a new value for {p.ParameterName} from the allowed
values {p.AllowedValues} ");

                   var choice = Console.ReadLine();
                   int.TryParse(choice, out newValue);
               }

               p.ParameterValue = newValue.ToString();
           }
       }
```

```
        var request = new ModifyDBClusterParameterGroupRequest
        {
            Parameters = parameters,
            DBClusterParameterGroupName = groupName,
        };

        var result = await _amazonRDS.ModifyDBClusterParameterGroupAsync(request);
        return result.DBClusterParameterGroupName;
    }
```

- Para obter detalhes da API, consulte Modificar DBCluster ParameterGroup na Referência AWS SDK para .NET da API.

Exemplos de Auto Scaling usando SDK para .NET (v4)

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET (v4) com Auto Scaling.

As noções básicas são exemplos de código que mostram como realizar as operações essenciais em um serviço.

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar perfis de serviço individuais, você pode ver as ações no contexto em seus cenários relacionados.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- Conceitos básicos
- Conceitos básicos
- Ações

Conceitos básicos

Olá, Auto Scaling

O exemplo de código a seguir mostra como começar a usar o Auto Scaling.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
namespace AutoScalingActions;

using Amazon.AutoScaling;

public class HelloAutoScaling
{
    /// <summary>
    /// Hello Amazon EC2 Auto Scaling. List EC2 Auto Scaling groups.
    /// </summary>
    /// <param name="args"></param>
    /// <returns>Async Task.</returns>
    static async Task Main(string[] args)
    {
        var client = new AmazonAutoScalingClient();

        Console.WriteLine("Welcome to Amazon EC2 Auto Scaling.");
        Console.WriteLine("Let's get a description of your Auto Scaling groups.");

        var response = await client.DescribeAutoScalingGroupsAsync();

        if (response.AutoScalingGroups == null || response.AutoScalingGroups.Count
 == 0)
        {
            Console.WriteLine("Sorry, you don't have any Amazon EC2 Auto Scaling
 groups.");
            return;
        }
```

```
        response.AutoScalingGroups.ForEach(autoScalingGroup =>
        {

 Console.WriteLine($"{autoScalingGroup.AutoScalingGroupName}\t{autoScalingGroup.Availability
        });

    }
}
```

- Para obter detalhes da API, consulte DescribeAutoScalingGroupsa Referência AWS SDK para .NET da API.

Conceitos básicos

Conheça os conceitos básicos

O exemplo de código a seguir mostra como:

- Criar um grupo do Amazon EC2 Auto Scaling com um modelo de inicialização e zonas de disponibilidade e obter informações sobre instâncias em execução.
- Ative a coleta de CloudWatch métricas da Amazon.
- Atualizar a capacidade desejada do grupo e aguardar a inicialização de uma instância.
- Encerrar uma instância no grupo.
- Listar as atividades de ajuste de escala que ocorrem em resposta às solicitações do usuário e às mudanças de capacidade.
- Obtenha estatísticas de CloudWatch métricas e, em seguida, limpe os recursos.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
global using Amazon.AutoScaling;
```

```
global using Amazon.AutoScaling.Model;
global using Amazon.CloudWatch;
global using AutoScalingActions;
global using Microsoft.Extensions.DependencyInjection;
global using Microsoft.Extensions.Hosting;
global using Microsoft.Extensions.Logging;
global using Microsoft.Extensions.Logging.Console;
global using Microsoft.Extensions.Logging.Debug;



using Amazon.EC2;
using Microsoft.Extensions.Configuration;
using Host = Microsoft.Extensions.Hosting.Host;

namespace AutoScalingBasics;

public class AutoScalingBasics
{

    static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon EC2 Auto Scaling, Amazon
        // CloudWatch, and Amazon EC2.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
 LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
            .ConfigureServices((_, services) =>
            services.AddAWSService<IAmazonAutoScaling>()
                .AddAWSService<IAmazonCloudWatch>()
                .AddAWSService<IAmazonEC2>()
                .AddTransient<AutoScalingWrapper>()
                .AddTransient<CloudWatchWrapper>()
                .AddTransient<EC2Wrapper>()
                .AddTransient<UIWrapper>()
            )
            .Build();


        var autoScalingWrapper =
 host.Services.GetRequiredService<AutoScalingWrapper>();
```

```
        var cloudWatchWrapper =
host.Services.GetRequiredService<CloudWatchWrapper>();
        var ec2Wrapper = host.Services.GetRequiredService<EC2Wrapper>();
        var uiWrapper = host.Services.GetRequiredService<UIWrapper>();

        var configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("settings.json") // Load test settings from .json file.
            .AddJsonFile("settings.local.json",
                true) // Optionally load local settings.
            .Build();

        var imageId = configuration["ImageId"];
        var instanceType = configuration["InstanceType"];
        var launchTemplateName = configuration["LaunchTemplateName"];

        launchTemplateName += Guid.NewGuid().ToString();

        // The name of the Auto Scaling group.
        var groupName = configuration["GroupName"];

        uiWrapper.DisplayTitle("Auto Scaling Basics");
        uiWrapper.DisplayAutoScalingBasicsDescription();

        // Create the launch template and save the template Id to use when deleting
the
        // launch template at the end of the application.
        var launchTemplateId = await ec2Wrapper.CreateLaunchTemplateAsync(imageId!,
instanceType!, launchTemplateName);

        // Confirm that the template was created by asking for a description of it.
        await ec2Wrapper.DescribeLaunchTemplateAsync(launchTemplateName);

        uiWrapper.PressEnter();

        var availabilityZones = await ec2Wrapper.ListAvailabilityZonesAsync();

        Console.WriteLine($"Creating an Auto Scaling group named {groupName}.");
        await autoScalingWrapper.CreateAutoScalingGroupAsync(
            groupName!,
            launchTemplateName,
            availabilityZones[0].ZoneName);

        // Keep checking the details of the new group until its lifecycle state
```

```
        // is "InService".
        Console.WriteLine($"Waiting for the Auto Scaling group to be active.");

        List<AutoScalingInstanceDetails> instanceDetails;

        do
        {
            instanceDetails = await
autoScalingWrapper.DescribeAutoScalingInstancesAsync(groupName!);
        }
        while (instanceDetails.Count <= 0);

        Console.WriteLine($"Auto scaling group {groupName} successfully created.");
        Console.WriteLine($"{instanceDetails.Count} instances were created for the
group.");

        // Display the details of the Auto Scaling group.
        instanceDetails.ForEach(detail =>
        {
            Console.WriteLine($"Group name: {detail.AutoScalingGroupName}");
        });

        uiWrapper.PressEnter();

        uiWrapper.DisplayTitle("Metrics collection");
        Console.WriteLine($"Enable metrics collection for {groupName}");
        await autoScalingWrapper.EnableMetricsCollectionAsync(groupName!);

        // Show the metrics that are collected for the group.

        // Update the maximum size of the group to three instances.
        Console.WriteLine("--- Update the Auto Scaling group to increase max size to
3 ---");
        int maxSize = 3;
        await autoScalingWrapper.UpdateAutoScalingGroupAsync(groupName!,
launchTemplateName, maxSize);

        Console.WriteLine("--- Describe all Auto Scaling groups to show the current
state of the group ---");
        var groups = await
autoScalingWrapper.DescribeAutoScalingGroupsAsync(groupName!);

        uiWrapper.DisplayGroupDetails(groups!);
```

```
        uiWrapper.PressEnter();

        uiWrapper.DisplayTitle("Describe account limits");
        await autoScalingWrapper.DescribeAccountLimitsAsync();

        uiWrapper.WaitABit(60, "Waiting for the resources to be ready.");

        uiWrapper.DisplayTitle("Set desired capacity");
        int desiredCapacity = 2;
        await autoScalingWrapper.SetDesiredCapacityAsync(groupName!,
desiredCapacity);

        Console.WriteLine("Get the two instance Id values");

        // Empty the group before getting the details again.
        groups.Clear();
        groups = await
autoScalingWrapper.DescribeAutoScalingGroupsAsync(groupName!);
        if (groups.Any())
        {
            foreach (AutoScalingGroup group in groups)
            {
                Console.WriteLine($"The group name is
{group.AutoScalingGroupName}");
                Console.WriteLine($"The group ARN is {group.AutoScalingGroupARN}");
                var instances = group.Instances;
                foreach (Amazon.AutoScaling.Model.Instance instance in instances)
                {
                    Console.WriteLine($"The instance id is {instance.InstanceId}");
                    Console.WriteLine($"The lifecycle state is
{instance.LifecycleState}");
                }
            }
        }

        uiWrapper.DisplayTitle("Scaling Activities");
        Console.WriteLine("Let's list the scaling activities that have occurred for
the group.");
        var activities = await
autoScalingWrapper.DescribeScalingActivitiesAsync(groupName!);
        if (activities.Any())
        {
            activities.ForEach(activity =>
            {
```

```
                Console.WriteLine($"The activity Id is {activity.ActivityId}");
                Console.WriteLine($"The activity details are {activity.Details}");
            });
        }

        // Display the Amazon CloudWatch metrics that have been collected.
        var metrics = await cloudWatchWrapper.GetCloudWatchMetricsAsync(groupName!);
        if (metrics.Any())
        {
            Console.WriteLine($"Metrics collected for {groupName}:");
            metrics.ForEach(metric =>
            {
                Console.Write($"Metric name: {metric.MetricName}\t");
                Console.WriteLine($"Namespace: {metric.Namespace}");
            });
        }

        var dataPoints = await
cloudWatchWrapper.GetMetricStatisticsAsync(groupName!);
        if (dataPoints.Any())
        {
            Console.WriteLine("Details for the metrics collected:");
            dataPoints.ForEach(detail => { Console.WriteLine(detail); });
        }

        // Disable metrics collection.
        Console.WriteLine("Disabling the collection of metrics for {groupName}.");
        var success = await
autoScalingWrapper.DisableMetricsCollectionAsync(groupName!);

        if (success)
        {
            Console.WriteLine($"Successfully stopped metrics collection for
{groupName}.");
        }
        else
        {
            Console.WriteLine($"Could not stop metrics collection for
{groupName}.");
        }

        // Terminate all instances in the group.
        uiWrapper.DisplayTitle("Terminating Auto Scaling instances");
```

```
        Console.WriteLine("Now terminating all instances in the Auto Scaling
 group.");

        if (groups is not null)
        {
            groups.ForEach(group =>
            {
                // Only delete instances in the AutoScaling group we created.
                if (group.AutoScalingGroupName == groupName)
                {
                    group.Instances.ForEach(async instance =>
                    {
                        await
 autoScalingWrapper.TerminateInstanceInAutoScalingGroupAsync(instance.InstanceId);
                    });
                }
            });
        }

        // After all instances are terminated, delete the group.
        uiWrapper.DisplayTitle("Clean up resources");
        Console.WriteLine("Deleting the Auto Scaling group.");
        await autoScalingWrapper.DeleteAutoScalingGroupAsync(groupName!);

        // Delete the launch template.
        var deletedLaunchTemplateName = await
 ec2Wrapper.DeleteLaunchTemplateAsync(launchTemplateId);

        if (deletedLaunchTemplateName == launchTemplateName)
        {
            Console.WriteLine("Successfully deleted the launch template.");
        }

        Console.WriteLine("The demo is now concluded.");
    }
}


namespace AutoScalingBasics;

/// <summary>
/// A class to provide user interface methods for the EC2 AutoScaling Basics
/// scenario.
/// </summary>
```

```
public class UIWrapper
{
    public readonly string SepBar = new('-', Console.WindowWidth);

    /// <summary>
    /// Describe the steps in the EC2 AutoScaling Basics scenario.
    /// </summary>
    public void DisplayAutoScalingBasicsDescription()
    {
        Console.WriteLine("This code example performs the following operations:");
        Console.WriteLine(" 1. Creates an Amazon EC2 launch template.");
        Console.WriteLine(" 2. Creates an Auto Scaling group.");
        Console.WriteLine(" 3. Shows the details of the new Auto Scaling group");
        Console.WriteLine("    to show that only one instance was created.");
        Console.WriteLine(" 4. Enables metrics collection.");
        Console.WriteLine(" 5. Updates the Auto Scaling group to increase the");
        Console.WriteLine("    capacity to three.");
        Console.WriteLine(" 6. Describes Auto Scaling groups again to show the");
        Console.WriteLine("    current state of the group.");
        Console.WriteLine(" 7. Changes the desired capacity of the Auto Scaling");
        Console.WriteLine("    group to use an additional instance.");
        Console.WriteLine(" 8. Shows that there are now instances in the group.");
        Console.WriteLine(" 9. Lists the scaling activities that have occurred for
 the group.");
        Console.WriteLine("10. Displays the Amazon CloudWatch metrics that have");
        Console.WriteLine("    been collected.");
        Console.WriteLine("11. Disables metrics collection.");
        Console.WriteLine("12. Terminates all instances in the Auto Scaling
 group.");
        Console.WriteLine("13. Deletes the Auto Scaling group.");
        Console.WriteLine("14. Deletes the Amazon EC2 launch template.");
        PressEnter();
    }

    /// <summary>
    /// Display information about the Amazon Ec2 AutoScaling groups passed
    /// in the list of AutoScalingGroup objects.
    /// </summary>
    /// <param name="groups">A list of AutoScalingGroup objects.</param>
    public void DisplayGroupDetails(List<AutoScalingGroup> groups)
    {
        if (groups is null)
            return;
```

```
        groups.ForEach(group =>
        {
            Console.WriteLine($"Group name:\t{group.AutoScalingGroupName}");
            Console.WriteLine($"Group created:\t{group.CreatedTime}");
            Console.WriteLine($"Maximum number of instances:\t{group.MaxSize}");
            Console.WriteLine($"Desired number of instances:
\t{group.DesiredCapacity}");
        });
    }

    /// <summary>
    /// Display a message and wait until the user presses enter.
    /// </summary>
    public void PressEnter()
    {
        Console.Write("\nPress <Enter> to continue. ");
        _ = Console.ReadLine();
        Console.WriteLine();
    }

    /// <summary>
    /// Pad a string with spaces to center it on the console display.
    /// </summary>
    /// <param name="strToCenter">The string to be centered.</param>
    /// <returns>The padded string.</returns>
    public string CenterString(string strToCenter)
    {
        var padAmount = (Console.WindowWidth - strToCenter.Length) / 2;
        var leftPad = new string(' ', padAmount);
        return $"{leftPad}{strToCenter}";
    }

    /// <summary>
    /// Display a line of hyphens, the centered text of the title and another
    /// line of hyphens.
    /// </summary>
    /// <param name="strTitle">The string to be displayed.</param>
    public void DisplayTitle(string strTitle)
    {
        Console.WriteLine(SepBar);
        Console.WriteLine(CenterString(strTitle));
        Console.WriteLine(SepBar);
    }
```

```
    /// <summary>
    /// Display a countdown and wait for a number of seconds.
    /// </summary>
    /// <param name="numSeconds">The number of seconds to wait.</param>
    public void WaitABit(int numSeconds, string msg)
    {
        Console.WriteLine(msg);

        // Wait for the requested number of seconds.
        for (int i = numSeconds; i > 0; i--)
        {
            System.Threading.Thread.Sleep(1000);
            Console.Write($"{i}...");
        }

        PressEnter();
    }
}
```

Defina as funções que são chamadas pelo cenário para gerenciar modelos e métricas de lançamento. Essas funções incluem Auto Scaling, Amazon EC2 e ações. CloudWatch

```
namespace AutoScalingActions;

using Amazon.AutoScaling;
using Amazon.AutoScaling.Model;

/// <summary>
/// A class that includes methods to perform Amazon EC2 Auto Scaling
/// actions.
/// </summary>
public class AutoScalingWrapper
{
    private readonly IAmazonAutoScaling _amazonAutoScaling;

    /// <summary>
    /// Constructor for the AutoScalingWrapper class.
    /// </summary>
    /// <param name="amazonAutoScaling">The injected Amazon EC2 Auto Scaling
 client.</param>
```

```
    public AutoScalingWrapper(IAmazonAutoScaling amazonAutoScaling)
    {
        _amazonAutoScaling = amazonAutoScaling;
    }


    /// <summary>
    /// Create a new Amazon EC2 Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name to use for the new Auto Scaling
    /// group.</param>
    /// <param name="launchTemplateName">The name of the Amazon EC2 Auto Scaling
    /// launch template to use to create instances in the group.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> CreateAutoScalingGroupAsync(
        string groupName,
        string launchTemplateName,
        string availabilityZone)
    {
        var templateSpecification = new LaunchTemplateSpecification
        {
            LaunchTemplateName = launchTemplateName,
        };

        var zoneList = new List<string>
            {
                availabilityZone,
            };

        var request = new CreateAutoScalingGroupRequest
        {
            AutoScalingGroupName = groupName,
            AvailabilityZones = zoneList,
            LaunchTemplate = templateSpecification,
            MaxSize = 6,
            MinSize = 1
        };
        try
        {
            var response = await
_amazonAutoScaling.CreateAutoScalingGroupAsync(request);
            Console.WriteLine($"{groupName} Auto Scaling Group created");
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
```

```
        catch (AlreadyExistsException)
        {
            Console.WriteLine($"{groupName} Auto Scaling Group already exists.");
            return true;
        }
    }



    /// <summary>
    /// Retrieve information about Amazon EC2 Auto Scaling quotas to the
    /// active AWS account.
    /// </summary>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DescribeAccountLimitsAsync()
    {
        var response = await _amazonAutoScaling.DescribeAccountLimitsAsync();
        Console.WriteLine("The maximum number of Auto Scaling groups is " +
 response.MaxNumberOfAutoScalingGroups);
        Console.WriteLine("The current number of Auto Scaling groups is " +
 response.NumberOfAutoScalingGroups);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }



    /// <summary>
    /// Retrieve a list of the Amazon EC2 Auto Scaling activities for an
    /// Amazon EC2 Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Amazon EC2 Auto Scaling group.</
param>
    /// <returns>A list of Amazon EC2 Auto Scaling activities.</returns>
    public async Task<List<Activity>> DescribeScalingActivitiesAsync(
        string groupName)
    {
        var activities = new List<Activity>();
        var scalingActivitiesRequest = new DescribeScalingActivitiesRequest
        {
            AutoScalingGroupName = groupName,
            MaxRecords = 10,
        };
```

```
        var response = await
 _amazonAutoScaling.DescribeScalingActivitiesAsync(scalingActivitiesRequest);
        if (response.Activities != null)
        {
            activities = response.Activities;
        }
        return activities;
    }



    /// <summary>
    /// Get data about the instances in an Amazon EC2 Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Amazon EC2 Auto Scaling group.</
param>
    /// <returns>A list of Amazon EC2 Auto Scaling details.</returns>
    public async Task<List<AutoScalingInstanceDetails>>
 DescribeAutoScalingInstancesAsync(
        string groupName)
    {
        var groups = await DescribeAutoScalingGroupsAsync(groupName);
        var instanceIds = new List<string>();
        var instanceDetails = new List<AutoScalingInstanceDetails>();
        if (groups != null)
        {
            groups.ForEach(group =>
            {
                if (group.AutoScalingGroupName == groupName && group.Instances !=
 null)
                {
                    group.Instances.ForEach(instance =>
                    {
                        instanceIds.Add(instance.InstanceId);
                    });
                }
            });

            var scalingGroupsRequest = new DescribeAutoScalingInstancesRequest
            {
                MaxRecords = 10,
                InstanceIds = instanceIds,
            };
```

```
            var response =
                await _amazonAutoScaling.DescribeAutoScalingInstancesAsync(
                    scalingGroupsRequest);
            if (response.AutoScalingInstances != null)
            {
                instanceDetails = response.AutoScalingInstances;
            }
        }

        return instanceDetails;
    }



    /// <summary>
    /// Retrieve a list of information about Amazon EC2 Auto Scaling groups.
    /// </summary>
    /// <param name="groupName">The name of the Amazon EC2 Auto Scaling group.</
param>
    /// <returns>A list of Amazon EC2 Auto Scaling groups.</returns>
    public async Task<List<AutoScalingGroup>> DescribeAutoScalingGroupsAsync(
        string groupName)
    {
        var groups = new List<AutoScalingGroup>();
        var groupList = new List<string>
            {
                groupName,
            };

        var request = new DescribeAutoScalingGroupsRequest
        {
            AutoScalingGroupNames = groupList,
        };

        var response = await
 _amazonAutoScaling.DescribeAutoScalingGroupsAsync(request);
        if (response.AutoScalingGroups != null)
        {
            groups = response.AutoScalingGroups;
        }

        return groups;
    }
```

```
    /// <summary>
    /// Delete an Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Amazon EC2 Auto Scaling group.</
param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteAutoScalingGroupAsync(
        string groupName)
    {
        var deleteAutoScalingGroupRequest = new DeleteAutoScalingGroupRequest
        {
            AutoScalingGroupName = groupName,
            ForceDelete = true,
        };

        var response = await
 _amazonAutoScaling.DeleteAutoScalingGroupAsync(deleteAutoScalingGroupRequest);
        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"You successfully deleted {groupName}");
            return true;
        }

        Console.WriteLine($"Couldn't delete {groupName}.");
        return false;
    }


    /// <summary>
    /// Disable the collection of metric data for an Amazon EC2 Auto Scaling
    /// group.
    /// </summary>
    /// <param name="groupName">The name of the Auto Scaling group.</param>
    /// <returns>A Boolean value that indicates the success or failure of
    /// the operation.</returns>
    public async Task<bool> DisableMetricsCollectionAsync(string groupName)
    {
        var request = new DisableMetricsCollectionRequest
        {
            AutoScalingGroupName = groupName,
        };
```

```
        var response = await
_amazonAutoScaling.DisableMetricsCollectionAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Enable the collection of metric data for an Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Auto Scaling group.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> EnableMetricsCollectionAsync(string groupName)
    {
        var listMetrics = new List<string>
            {
                "GroupMaxSize",
            };

        var collectionRequest = new EnableMetricsCollectionRequest
        {
            AutoScalingGroupName = groupName,
            Metrics = listMetrics,
            Granularity = "1Minute",
        };

        var response = await
_amazonAutoScaling.EnableMetricsCollectionAsync(collectionRequest);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Set the desired capacity of an Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Auto Scaling group.</param>
    /// <param name="desiredCapacity">The desired capacity for the Auto
    /// Scaling group.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> SetDesiredCapacityAsync(
        string groupName,
        int desiredCapacity)
    {
        var capacityRequest = new SetDesiredCapacityRequest
        {
```

```
            AutoScalingGroupName = groupName,
            DesiredCapacity = desiredCapacity,
        };

        var response = await
 _amazonAutoScaling.SetDesiredCapacityAsync(capacityRequest);
        Console.WriteLine($"You have set the DesiredCapacity to
 {desiredCapacity}.");

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }


    /// <summary>
    /// Terminate all instances in the Auto Scaling group in preparation for
    /// deleting the group.
    /// </summary>
    /// <param name="instanceId">The instance Id of the instance to terminate.</
param>
    /// <returns>A Boolean value that indicates the success or failure of
    /// the operation.</returns>
    public async Task<bool> TerminateInstanceInAutoScalingGroupAsync(
        string instanceId)
    {
        var request = new TerminateInstanceInAutoScalingGroupRequest
        {
            InstanceId = instanceId,
            ShouldDecrementDesiredCapacity = false,
        };

        var response = await
 _amazonAutoScaling.TerminateInstanceInAutoScalingGroupAsync(request);

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"You have terminated the instance: {instanceId}");
            return true;
        }

        Console.WriteLine($"Could not terminate {instanceId}");
        return false;
    }
```

```
    /// <summary>
    /// Update the capacity of an Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Auto Scaling group.</param>
    /// <param name="launchTemplateName">The name of the EC2 launch template.</
param>
    /// <param name="maxSize">The maximum number of instances that can be
    /// created for the Auto Scaling group.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> UpdateAutoScalingGroupAsync(
        string groupName,
        string launchTemplateName,
        int maxSize)
    {
        var templateSpecification = new LaunchTemplateSpecification
        {
            LaunchTemplateName = launchTemplateName,
        };

        var groupRequest = new UpdateAutoScalingGroupRequest
        {
            MaxSize = maxSize,
            AutoScalingGroupName = groupName,
            LaunchTemplate = templateSpecification,
        };

        var response = await
 _amazonAutoScaling.UpdateAutoScalingGroupAsync(groupRequest);
        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"You successfully updated the Auto Scaling group
 {groupName}.");
            return true;
        }
        else
        {
            return false;
        }
    }

}


namespace AutoScalingActions;
```

```
using Amazon.EC2;
using Amazon.EC2.Model;

public class EC2Wrapper
{
    private readonly IAmazonEC2 _amazonEc2;

    /// <summary>
    /// Constructor for the EC2Wrapper class.
    /// </summary>
    /// <param name="amazonEc2">The injected Amazon EC2 client.</param>
    public EC2Wrapper(IAmazonEC2 amazonEc2)
    {
        _amazonEc2 = amazonEc2;
    }

    /// <summary>
    /// Create a new Amazon EC2 launch template.
    /// </summary>
    /// <param name="imageId">The image Id to use for instances launched
    /// using the Amazon EC2 launch template.</param>
    /// <param name="instanceType">The type of EC2 instances to create.</param>
    /// <param name="launchTemplateName">The name of the launch template.</param>
    /// <returns>Returns the TemplateID of the new launch template.</returns>
    public async Task<string> CreateLaunchTemplateAsync(
        string imageId,
        string instanceType,
        string launchTemplateName)
    {
        var request = new CreateLaunchTemplateRequest
        {
            LaunchTemplateData = new RequestLaunchTemplateData
            {
                ImageId = imageId,
                InstanceType = instanceType,
            },
            LaunchTemplateName = launchTemplateName,
        };

        var response = await _amazonEc2.CreateLaunchTemplateAsync(request);

        return response.LaunchTemplate.LaunchTemplateId;
    }
```

```
    /// <summary>
    /// Delete an Amazon EC2 launch template.
    /// </summary>
    /// <param name="launchTemplateId">The TemplateId of the launch template to
    /// delete.</param>
    /// <returns>The name of the EC2 launch template that was deleted.</returns>
    public async Task<string> DeleteLaunchTemplateAsync(string launchTemplateId)
    {
        var request = new DeleteLaunchTemplateRequest
        {
            LaunchTemplateId = launchTemplateId,
        };

        var response = await _amazonEc2.DeleteLaunchTemplateAsync(request);
        return response.LaunchTemplate.LaunchTemplateName;
    }

    /// <summary>
    /// Retrieve information about an EC2 launch template.
    /// </summary>
    /// <param name="launchTemplateName">The name of the EC2 launch template.</
param>
    /// <returns>A Boolean value that indicates the success or failure of
    /// the operation.</returns>
    public async Task<bool> DescribeLaunchTemplateAsync(string launchTemplateName)
    {
        var request = new DescribeLaunchTemplatesRequest
        {
            LaunchTemplateNames = new List<string> { launchTemplateName, },
        };

        var response = await _amazonEc2.DescribeLaunchTemplatesAsync(request);

        if (response.LaunchTemplates is not null)
        {
            response.LaunchTemplates.ForEach(template =>
            {
                Console.Write($"{template.LaunchTemplateName}\t");
                Console.WriteLine(template.LaunchTemplateId);
            });

            return true;
        }
```

```
        return false;
    }

    /// <summary>
    /// Retrieve the availability zones for the current region.
    /// </summary>
    /// <returns>A collection of availability zones.</returns>
    public async Task<List<AvailabilityZone>> ListAvailabilityZonesAsync()
    {
        var response = await _amazonEc2.DescribeAvailabilityZonesAsync(
            new DescribeAvailabilityZonesRequest());

        return response.AvailabilityZones;
    }
}


namespace AutoScalingActions;

using Amazon.CloudWatch;
using Amazon.CloudWatch.Model;

/// <summary>
/// Contains methods to access Amazon CloudWatch metrics for the
/// Amazon EC2 Auto Scaling basics scenario.
/// </summary>
public class CloudWatchWrapper
{
    private readonly IAmazonCloudWatch _amazonCloudWatch;

    /// <summary>
    /// Constructor for the CloudWatchWrapper.
    /// </summary>
    /// <param name="amazonCloudWatch">The injected CloudWatch client.</param>
    public CloudWatchWrapper(IAmazonCloudWatch amazonCloudWatch)
    {
        _amazonCloudWatch = amazonCloudWatch;
    }

    /// <summary>
    /// Retrieve the metrics information collection for the Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Auto Scaling group.</param>
```

```
    /// <returns>A list of Metrics collected for the Auto Scaling group.</returns>
    public async Task<List<Metric>> GetCloudWatchMetricsAsync(string groupName)
    {
        var metrics = new List<Metric>();
        var filter = new DimensionFilter
        {
            Name = "AutoScalingGroupName",
            Value = $"{groupName}",
        };

        var request = new ListMetricsRequest
        {
            MetricName = "AutoScalingGroupName",
            Dimensions = new List<DimensionFilter> { filter },
            Namespace = "AWS/AutoScaling",
        };

        var response = await _amazonCloudWatch.ListMetricsAsync(request);
        if (response.Metrics != null)
        {
            metrics = response.Metrics;
        }
        return metrics;
    }

    /// <summary>
    /// Retrieve the metric data collected for an Amazon EC2 Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Amazon EC2 Auto Scaling group.</
param>
    /// <returns>A list of data points.</returns>
    public async Task<List<Datapoint>> GetMetricStatisticsAsync(string groupName)
    {
        var dataPoints = new List<Datapoint>();
        var metricDimensions = new List<Dimension>
            {
                new Dimension
                {
                    Name = "AutoScalingGroupName",
                    Value = $"{groupName}",
                },
            };

        // The start time will be yesterday.
```

```
        var startTime = DateTime.UtcNow.AddDays(-1);

        var request = new GetMetricStatisticsRequest
        {
            MetricName = "AutoScalingGroupName",
            Dimensions = metricDimensions,
            Namespace = "AWS/AutoScaling",
            Period = 60, // 60 seconds.
            Statistics = new List<string>() { "Minimum" },
            StartTimeUtc = startTime,
            EndTimeUtc = DateTime.UtcNow,
        };

        var response = await _amazonCloudWatch.GetMetricStatisticsAsync(request);
        if (response.Datapoints != null)
        {
            dataPoints = response.Datapoints;
        }

        return dataPoints;
    }

}
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para .NET .
  - CreateAutoScalingGroup
  - DeleteAutoScalingGroup
  - DescribeAutoScalingGroups
  - DescribeAutoScalingInstances
  - DescribeScalingActivities
  - DisableMetricsCollection
  - EnableMetricsCollection
  - SetDesiredCapacity
  - TerminateInstanceInAutoScalingGroup
  - UpdateAutoScalingGroup

Ações

`CreateAutoScalingGroup`

O código de exemplo a seguir mostra como usar `CreateAutoScalingGroup`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
/// <summary>
/// Create a new Amazon EC2 Auto Scaling group.
/// </summary>
/// <param name="groupName">The name to use for the new Auto Scaling
/// group.</param>
/// <param name="launchTemplateName">The name of the Amazon EC2 Auto Scaling
/// launch template to use to create instances in the group.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> CreateAutoScalingGroupAsync(
    string groupName,
    string launchTemplateName,
    string availabilityZone)
{
    var templateSpecification = new LaunchTemplateSpecification
    {
        LaunchTemplateName = launchTemplateName,
    };

    var zoneList = new List<string>
        {
            availabilityZone,
        };

    var request = new CreateAutoScalingGroupRequest
    {
        AutoScalingGroupName = groupName,
        AvailabilityZones = zoneList,
```

```
            LaunchTemplate = templateSpecification,
            MaxSize = 6,
            MinSize = 1
        };
        try
        {
            var response = await
_amazonAutoScaling.CreateAutoScalingGroupAsync(request);
            Console.WriteLine($"{groupName} Auto Scaling Group created");
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AlreadyExistsException)
        {
            Console.WriteLine($"{groupName} Auto Scaling Group already exists.");
            return true;
        }
    }
```

- Para obter detalhes da API, consulte CreateAutoScalingGroupa Referência AWS SDK para .NET da API.

`DescribeAutoScalingGroups`

O código de exemplo a seguir mostra como usar `DescribeAutoScalingGroups`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Get data about the instances in an Amazon EC2 Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Amazon EC2 Auto Scaling group.</
param>
```

```
    /// <returns>A list of Amazon EC2 Auto Scaling details.</returns>
    public async Task<List<AutoScalingInstanceDetails>>
DescribeAutoScalingInstancesAsync(
        string groupName)
    {
        var groups = await DescribeAutoScalingGroupsAsync(groupName);
        var instanceIds = new List<string>();
        var instanceDetails = new List<AutoScalingInstanceDetails>();
        if (groups != null)
        {
            groups.ForEach(group =>
            {
                if (group.AutoScalingGroupName == groupName && group.Instances !=
null)
                {
                    group.Instances.ForEach(instance =>
                    {
                        instanceIds.Add(instance.InstanceId);
                    });
                }
            });

            var scalingGroupsRequest = new DescribeAutoScalingInstancesRequest
            {
                MaxRecords = 10,
                InstanceIds = instanceIds,
            };

            var response =
                await _amazonAutoScaling.DescribeAutoScalingInstancesAsync(
                    scalingGroupsRequest);
            if (response.AutoScalingInstances != null)
            {
                instanceDetails = response.AutoScalingInstances;
            }
        }

        return instanceDetails;
    }
```

- Para obter detalhes da API, consulte DescribeAutoScalingGroupsa Referência AWS SDK para .NET da API.

`DescribeAutoScalingInstances`

O código de exemplo a seguir mostra como usar `DescribeAutoScalingInstances`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Get data about the instances in an Amazon EC2 Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Amazon EC2 Auto Scaling group.</
param>
    /// <returns>A list of Amazon EC2 Auto Scaling details.</returns>
    public async Task<List<AutoScalingInstanceDetails>>
 DescribeAutoScalingInstancesAsync(
        string groupName)
    {
        var groups = await DescribeAutoScalingGroupsAsync(groupName);
        var instanceIds = new List<string>();
        var instanceDetails = new List<AutoScalingInstanceDetails>();
        if (groups != null)
        {
            groups.ForEach(group =>
            {
                if (group.AutoScalingGroupName == groupName && group.Instances !=
 null)
                {
                    group.Instances.ForEach(instance =>
                    {
                        instanceIds.Add(instance.InstanceId);
                    });
                }
            });
```

```
            var scalingGroupsRequest = new DescribeAutoScalingInstancesRequest
            {
                MaxRecords = 10,
                InstanceIds = instanceIds,
            };

            var response =
                await _amazonAutoScaling.DescribeAutoScalingInstancesAsync(
                    scalingGroupsRequest);
            if (response.AutoScalingInstances != null)
            {
                instanceDetails = response.AutoScalingInstances;
            }
        }

        return instanceDetails;
    }
```

- Para obter detalhes da API, consulte DescribeAutoScalingInstancesa Referência AWS SDK para .NET da API.

`DescribeScalingActivities`

O código de exemplo a seguir mostra como usar `DescribeScalingActivities`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Retrieve a list of the Amazon EC2 Auto Scaling activities for an
    /// Amazon EC2 Auto Scaling group.
    /// </summary>
```

```
    /// <param name="groupName">The name of the Amazon EC2 Auto Scaling group.</
param>
    /// <returns>A list of Amazon EC2 Auto Scaling activities.</returns>
    public async Task<List<Activity>> DescribeScalingActivitiesAsync(
        string groupName)
    {
        var activities = new List<Activity>();
        var scalingActivitiesRequest = new DescribeScalingActivitiesRequest
        {
            AutoScalingGroupName = groupName,
            MaxRecords = 10,
        };

        var response = await
 _amazonAutoScaling.DescribeScalingActivitiesAsync(scalingActivitiesRequest);
        if (response.Activities != null)
        {
            activities = response.Activities;
        }
        return activities;
    }
```

- Para obter detalhes da API, consulte DescribeScalingActivitiesa Referência AWS SDK para .NET da API.

`DisableMetricsCollection`

O código de exemplo a seguir mostra como usar `DisableMetricsCollection`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Disable the collection of metric data for an Amazon EC2 Auto Scaling
```

```
    /// group.
    /// </summary>
    /// <param name="groupName">The name of the Auto Scaling group.</param>
    /// <returns>A Boolean value that indicates the success or failure of
    /// the operation.</returns>
    public async Task<bool> DisableMetricsCollectionAsync(string groupName)
    {
        var request = new DisableMetricsCollectionRequest
        {
            AutoScalingGroupName = groupName,
        };

        var response = await
_amazonAutoScaling.DisableMetricsCollectionAsync(request);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- Para obter detalhes da API, consulte DisableMetricsCollectiona Referência AWS SDK para .NET da API.

`EnableMetricsCollection`

O código de exemplo a seguir mostra como usar `EnableMetricsCollection`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Enable the collection of metric data for an Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Auto Scaling group.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> EnableMetricsCollectionAsync(string groupName)
    {
```

```
        var listMetrics = new List<string>
            {
                "GroupMaxSize",
            };

        var collectionRequest = new EnableMetricsCollectionRequest
        {
            AutoScalingGroupName = groupName,
            Metrics = listMetrics,
            Granularity = "1Minute",
        };

        var response = await
_amazonAutoScaling.EnableMetricsCollectionAsync(collectionRequest);
        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- Para obter detalhes da API, consulte EnableMetricsCollectiona Referência AWS SDK para .NET da API.

`SetDesiredCapacity`

O código de exemplo a seguir mostra como usar `SetDesiredCapacity`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Set the desired capacity of an Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Auto Scaling group.</param>
    /// <param name="desiredCapacity">The desired capacity for the Auto
    /// Scaling group.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
```

```
    public async Task<bool> SetDesiredCapacityAsync(
        string groupName,
        int desiredCapacity)
    {
        var capacityRequest = new SetDesiredCapacityRequest
        {
            AutoScalingGroupName = groupName,
            DesiredCapacity = desiredCapacity,
        };

        var response = await
_amazonAutoScaling.SetDesiredCapacityAsync(capacityRequest);
        Console.WriteLine($"You have set the DesiredCapacity to
{desiredCapacity}.");

        return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
    }
```

- Para obter detalhes da API, consulte SetDesiredCapacitya Referência AWS SDK para .NET da API.

`TerminateInstanceInAutoScalingGroup`

O código de exemplo a seguir mostra como usar `TerminateInstanceInAutoScalingGroup`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Terminate all instances in the Auto Scaling group in preparation for
    /// deleting the group.
    /// </summary>
    /// <param name="instanceId">The instance Id of the instance to terminate.</
param>
```

```
    /// <returns>A Boolean value that indicates the success or failure of
    /// the operation.</returns>
    public async Task<bool> TerminateInstanceInAutoScalingGroupAsync(
        string instanceId)
    {
        var request = new TerminateInstanceInAutoScalingGroupRequest
        {
            InstanceId = instanceId,
            ShouldDecrementDesiredCapacity = false,
        };

        var response = await
_amazonAutoScaling.TerminateInstanceInAutoScalingGroupAsync(request);

        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"You have terminated the instance: {instanceId}");
            return true;
        }

        Console.WriteLine($"Could not terminate {instanceId}");
        return false;
    }
```

- Para obter detalhes da API, consulte TerminateInstanceInAutoScalingGroupa Referência AWS SDK para .NET da API.

`UpdateAutoScalingGroup`

O código de exemplo a seguir mostra como usar `UpdateAutoScalingGroup`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Update the capacity of an Auto Scaling group.
    /// </summary>
    /// <param name="groupName">The name of the Auto Scaling group.</param>
    /// <param name="launchTemplateName">The name of the EC2 launch template.</
param>
    /// <param name="maxSize">The maximum number of instances that can be
    /// created for the Auto Scaling group.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> UpdateAutoScalingGroupAsync(
        string groupName,
        string launchTemplateName,
        int maxSize)
    {
        var templateSpecification = new LaunchTemplateSpecification
        {
            LaunchTemplateName = launchTemplateName,
        };

        var groupRequest = new UpdateAutoScalingGroupRequest
        {
            MaxSize = maxSize,
            AutoScalingGroupName = groupName,
            LaunchTemplate = templateSpecification,
        };

        var response = await
 _amazonAutoScaling.UpdateAutoScalingGroupAsync(groupRequest);
        if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
        {
            Console.WriteLine($"You successfully updated the Auto Scaling group
 {groupName}.");
            return true;
        }
        else
        {
            return false;
        }
    }
```

- Para obter detalhes da API, consulte UpdateAutoScalingGroupa Referência AWS SDK para .NET da API.

Exemplos do Amazon Bedrock usando SDK para .NET (v4)

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET (v4) com o Amazon Bedrock.

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar perfis de serviço individuais, você pode ver as ações no contexto em seus cenários relacionados.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- Conceitos básicos
- Ações

Conceitos básicos

Olá, Amazon Bedrock

O exemplo de código a seguir mostra como começar a usar o Amazon Bedrock.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
using Amazon;
using Amazon.Bedrock;
using Amazon.Bedrock.Model;

namespace BedrockActions;
```

```
/// <summary>
/// This example shows how to list foundation models.
/// </summary>
internal class HelloBedrock
{
    /// <summary>
    /// Main method to call the ListFoundationModelsAsync method.
    /// </summary>
    /// <param name="args"> The command line arguments. </param>
    static async Task Main(string[] args)
    {
        // Specify a region endpoint where Amazon Bedrock is available. For a list
 of supported region see https://docs.aws.amazon.com/bedrock/latest/userguide/what-
is-bedrock.html#bedrock-regions
        AmazonBedrockClient bedrockClient = new(RegionEndpoint.USWest2);

        await ListFoundationModelsAsync(bedrockClient);

    }


    /// <summary>
    /// List foundation models.
    /// </summary>
    /// <param name="bedrockClient"> The Amazon Bedrock client. </param>
    private static async Task ListFoundationModelsAsync(AmazonBedrockClient
 bedrockClient)
    {
        Console.WriteLine("List foundation models with no filter.");

        try
        {
            var response = await bedrockClient.ListFoundationModelsAsync(new
 ListFoundationModelsRequest()
            {
            });

            if (response?.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                foreach (var fm in response.ModelSummaries)
                {
                    WriteToConsole(fm);
                }
            }
```

```
            else
            {
                Console.WriteLine("Something wrong happened");
            }
        }
        catch (AmazonBedrockException e)
        {
            Console.WriteLine(e.Message);
        }
    }


    /// <summary>
    /// Write the foundation model summary to console.
    /// </summary>
    /// <param name="foundationModel"> The foundation model summary to write to
 console. </param>
    private static void WriteToConsole(FoundationModelSummary foundationModel)
    {
        Console.WriteLine($"{foundationModel.ModelId}, Customization:
 {string.Join(", ", foundationModel.CustomizationsSupported)}, Stream:
 {foundationModel.ResponseStreamingSupported}, Input: {string.Join(",
 ", foundationModel.InputModalities)}, Output: {string.Join(", ",
 foundationModel.OutputModalities)}");
    }
}
```

- Para obter detalhes da API, consulte ListFoundationModelsa Referência AWS SDK para .NET da API.

Ações

`ListFoundationModels`

O código de exemplo a seguir mostra como usar `ListFoundationModels`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Listar os modelos de base do Bedrock disponíveis.

```
   /// <summary>
   /// List foundation models.
   /// </summary>
   /// <param name="bedrockClient"> The Amazon Bedrock client. </param>
   private static async Task ListFoundationModelsAsync(AmazonBedrockClient
bedrockClient)
   {
       Console.WriteLine("List foundation models with no filter.");

       try
       {
           var response = await bedrockClient.ListFoundationModelsAsync(new
ListFoundationModelsRequest()
           {
           });

           if (response?.HttpStatusCode == System.Net.HttpStatusCode.OK)
           {
               foreach (var fm in response.ModelSummaries)
               {
                   WriteToConsole(fm);
               }
           }
           else
           {
               Console.WriteLine("Something wrong happened");
           }
       }
       catch (AmazonBedrockException e)
       {
           Console.WriteLine(e.Message);
       }
```

```
    }
```

- Para obter detalhes da API, consulte ListFoundationModelsa Referência AWS SDK para .NET da API.

Exemplos de Amazon Bedrock Runtime usando SDK para .NET (v4)

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET (v4) com o Amazon Bedrock Runtime.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- Claude da Anthropic
- Command da Cohere
- Llama da Meta
- Mistral AI

Claude da Anthropic

Converse

O exemplo de código a seguir mostra como enviar uma mensagem de texto ao Claude da Anthropic usando a API Converse do Bedrock.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Envie uma mensagem de texto ao Claude da Anthropic usando a API Converse do Bedrock.

```
// Use the Converse API to send a text message to Anthropic Claude.

using System;
using System.Collections.Generic;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Claude 3 Haiku.
var modelId = "anthropic.claude-3-haiku-20240307-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

// Create a request with the model ID, the user message, and an inference
 configuration.
var request = new ConverseRequest
{
    ModelId = modelId,
    Messages = new List<Message>
    {
        new Message
        {
            Role = ConversationRole.User,
            Content = new List<ContentBlock> { new ContentBlock { Text =
 userMessage } }
        }
    },
    InferenceConfig = new InferenceConfiguration()
    {
        MaxTokens = 512,
        Temperature = 0.5F,
        TopP = 0.9F
    }
};

try
{
    // Send the request to the Bedrock Runtime and wait for the result.
```

```
    var response = await client.ConverseAsync(request);

    // Extract and print the response text.
    string responseText = response?.Output?.Message?.Content?[0]?.Text ?? "";
    Console.WriteLine(responseText);
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Consulte detalhes da API em Converse na Referência de API do AWS SDK para .NET .

ConverseStream

O exemplo de código a seguir mostra como enviar uma mensagem de texto ao Claude da Anthropic usando a API Converse do Bedrock e processar o fluxo de resposta em tempo real.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Envie uma mensagem de texto ao Claude da Anthropic usando a API Converse do Bedrock e processe o fluxo de resposta em tempo real.

```
// Use the Converse API to send a text message to Anthropic Claude
// and print the response stream.

using System;
using System.Collections.Generic;
using System.Linq;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;
```

```
// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Claude 3 Haiku.
var modelId = "anthropic.claude-3-haiku-20240307-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

// Create a request with the model ID, the user message, and an inference
 configuration.
var request = new ConverseStreamRequest
{
    ModelId = modelId,
    Messages = new List<Message>
    {
        new Message
        {
            Role = ConversationRole.User,
            Content = new List<ContentBlock> { new ContentBlock { Text =
 userMessage } }
        }
    },
    InferenceConfig = new InferenceConfiguration()
    {
        MaxTokens = 512,
        Temperature = 0.5F,
        TopP = 0.9F
    }
};

try
{
    // Send the request to the Bedrock Runtime and wait for the result.
    var response = await client.ConverseStreamAsync(request);

    // Extract and print the streamed response text in real-time.
    foreach (var chunk in response.Stream.AsEnumerable())
    {
        if (chunk is ContentBlockDeltaEvent)
        {
            Console.Write((chunk as ContentBlockDeltaEvent).Delta.Text);
        }
    }
```

```
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Para obter detalhes da API, consulte ConverseStreama Referência AWS SDK para .NET da API.

InvokeModel

O exemplo de código a seguir mostra como enviar uma mensagem de texto ao Claude da Anthropic usando a API Invoke Model.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Use a API InvokeModel para enviar uma mensagem de texto.

```
// Use the native inference API to send a text message to Anthropic Claude.

using System;
using System.IO;
using System.Text.Json;
using System.Text.Json.Nodes;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Claude 3 Haiku.
var modelId = "anthropic.claude-3-haiku-20240307-v1:0";
```

```
// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

//Format the request payload using the model's native structure.
var nativeRequest = JsonSerializer.Serialize(new
{
    anthropic_version = "bedrock-2023-05-31",
    max_tokens = 512,
    temperature = 0.5,
    messages = new[]
    {
        new { role = "user", content = userMessage }
    }
});

// Create a request with the model ID and the model's native request payload.
var request = new InvokeModelRequest()
{
    ModelId = modelId,
    Body = new MemoryStream(System.Text.Encoding.UTF8.GetBytes(nativeRequest)),
    ContentType = "application/json"
};

try
{
    // Send the request to the Bedrock Runtime and wait for the response.
    var response = await client.InvokeModelAsync(request);

    // Decode the response body.
    var modelResponse = await JsonNode.ParseAsync(response.Body);

    // Extract and print the response text.
    var responseText = modelResponse["content"]?[0]?["text"] ?? "";
    Console.WriteLine(responseText);
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Para obter detalhes da API, consulte InvokeModela Referência AWS SDK para .NET da API.

Command da Cohere

Converse

O exemplo de código a seguir mostra como enviar uma mensagem de texto ao Command da Cohere usando a API Converse do Bedrock.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Envie uma mensagem de texto ao Cohere Command usando a API Converse do Bedrock.

```
// Use the Converse API to send a text message to Cohere Command.

using System;
using System.Collections.Generic;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Command R.
var modelId = "cohere.command-r-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

// Create a request with the model ID, the user message, and an inference
 configuration.
var request = new ConverseRequest
{
    ModelId = modelId,
```

```
    Messages = new List<Message>
    {
        new Message
        {
            Role = ConversationRole.User,
            Content = new List<ContentBlock> { new ContentBlock { Text =
 userMessage } }
        }
    },
    InferenceConfig = new InferenceConfiguration()
    {
        MaxTokens = 512,
        Temperature = 0.5F,
        TopP = 0.9F
    }
};

try
{
    // Send the request to the Bedrock Runtime and wait for the result.
    var response = await client.ConverseAsync(request);

    // Extract and print the response text.
    string responseText = response?.Output?.Message?.Content?[0]?.Text ?? "";
    Console.WriteLine(responseText);
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Consulte detalhes da API em Converse na Referência de API do AWS SDK para .NET .

ConverseStream

O exemplo de código a seguir mostra como enviar uma mensagem de texto ao Command da Cohere usando a API Converse do Bedrock e processar o fluxo de resposta em tempo real.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Envie uma mensagem de texto ao Command da Cohere usando a API Converse do Bedrock e processe o fluxo de resposta em tempo real.

```
// Use the Converse API to send a text message to Cohere Command
// and print the response stream.

using System;
using System.Collections.Generic;
using System.Linq;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Command R.
var modelId = "cohere.command-r-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

// Create a request with the model ID, the user message, and an inference
 configuration.
var request = new ConverseStreamRequest
{
    ModelId = modelId,
    Messages = new List<Message>
    {
        new Message
        {
            Role = ConversationRole.User,
            Content = new List<ContentBlock> { new ContentBlock { Text =
 userMessage } }
```

```
        }
    },
    InferenceConfig = new InferenceConfiguration()
    {
        MaxTokens = 512,
        Temperature = 0.5F,
        TopP = 0.9F
    }
};

try
{
    // Send the request to the Bedrock Runtime and wait for the result.
    var response = await client.ConverseStreamAsync(request);

    // Extract and print the streamed response text in real-time.
    foreach (var chunk in response.Stream.AsEnumerable())
    {
        if (chunk is ContentBlockDeltaEvent)
        {
            Console.Write((chunk as ContentBlockDeltaEvent).Delta.Text);
        }
    }
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Para obter detalhes da API, consulte ConverseStreama Referência AWS SDK para .NET da API.

Llama da Meta

Converse

O exemplo de código a seguir mostra como enviar uma mensagem de texto ao Llama da Meta usando a API Converse do Bedrock.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Envie uma mensagem de texto ao Llama da Meta usando a API Converse do Bedrock.

```
// Use the Converse API to send a text message to Meta Llama.

using System;
using System.Collections.Generic;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Llama 3 8b Instruct.
var modelId = "meta.llama3-8b-instruct-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

// Create a request with the model ID, the user message, and an inference
 configuration.
var request = new ConverseRequest
{
    ModelId = modelId,
    Messages = new List<Message>
    {
        new Message
        {
            Role = ConversationRole.User,
            Content = new List<ContentBlock> { new ContentBlock { Text =
 userMessage } }
        }
    },
    InferenceConfig = new InferenceConfiguration()
    {
```

```
        MaxTokens = 512,
        Temperature = 0.5F,
        TopP = 0.9F
    }
};

try
{
    // Send the request to the Bedrock Runtime and wait for the result.
    var response = await client.ConverseAsync(request);

    // Extract and print the response text.
    string responseText = response?.Output?.Message?.Content?[0]?.Text ?? "";
    Console.WriteLine(responseText);
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Consulte detalhes da API em Converse na Referência de API do AWS SDK para .NET .

ConverseStream

O exemplo de código a seguir mostra como enviar uma mensagem de texto ao Llama da Meta usando a API Converse do Bedrock e processar o fluxo de resposta em tempo real.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Envie uma mensagem de texto ao Llama da Meta usando a API Converse do Bedrock e processe o fluxo de resposta em tempo real.

```
// Use the Converse API to send a text message to Meta Llama
```

```
// and print the response stream.

using System;
using System.Collections.Generic;
using System.Linq;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Llama 3 8b Instruct.
var modelId = "meta.llama3-8b-instruct-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

// Create a request with the model ID, the user message, and an inference
 configuration.
var request = new ConverseStreamRequest
{
    ModelId = modelId,
    Messages = new List<Message>
    {
        new Message
        {
            Role = ConversationRole.User,
            Content = new List<ContentBlock> { new ContentBlock { Text =
 userMessage } }
        }
    },
    InferenceConfig = new InferenceConfiguration()
    {
        MaxTokens = 512,
        Temperature = 0.5F,
        TopP = 0.9F
    }
};

try
{
    // Send the request to the Bedrock Runtime and wait for the result.
    var response = await client.ConverseStreamAsync(request);
```

```
    // Extract and print the streamed response text in real-time.
    foreach (var chunk in response.Stream.AsEnumerable())
    {
        if (chunk is ContentBlockDeltaEvent)
        {
            Console.Write((chunk as ContentBlockDeltaEvent).Delta.Text);
        }
    }
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Para obter detalhes da API, consulte ConverseStreama Referência AWS SDK para .NET da API.

Mistral AI

Converse

O exemplo de código a seguir mostra como enviar uma mensagem de texto à Mistral usando a API Converse do Bedrock.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Envie uma mensagem de texto à Mistral usando a API Converse do Bedrock.

```
// Use the Converse API to send a text message to Mistral.

using System;
```

```
using System.Collections.Generic;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Mistral Large.
var modelId = "mistral.mistral-large-2402-v1:0";

// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

// Create a request with the model ID, the user message, and an inference
 configuration.
var request = new ConverseRequest
{
    ModelId = modelId,
    Messages = new List<Message>
    {
        new Message
        {
            Role = ConversationRole.User,
            Content = new List<ContentBlock> { new ContentBlock { Text =
 userMessage } }
        }
    },
    InferenceConfig = new InferenceConfiguration()
    {
        MaxTokens = 512,
        Temperature = 0.5F,
        TopP = 0.9F
    }
};

try
{
    // Send the request to the Bedrock Runtime and wait for the result.
    var response = await client.ConverseAsync(request);

    // Extract and print the response text.
    string responseText = response?.Output?.Message?.Content?[0]?.Text ?? "";
    Console.WriteLine(responseText);
```

```
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
}
```

- Para obter detalhes da API, consulte Converse na Referência da API do AWS SDK para .NET .

ConverseStream

O exemplo de código a seguir mostra como enviar uma mensagem de texto à Mistral usando a API Converse do Bedrock e processar o fluxo de resposta em tempo real.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Envie uma mensagem de texto para a Mistral usando a API Converse do Bedrock e processe o fluxo de resposta em tempo real.

```
// Use the Converse API to send a text message to Mistral
// and print the response stream.

using System;
using System.Collections.Generic;
using System.Linq;
using Amazon;
using Amazon.BedrockRuntime;
using Amazon.BedrockRuntime.Model;

// Create a Bedrock Runtime client in the AWS Region you want to use.
var client = new AmazonBedrockRuntimeClient(RegionEndpoint.USEast1);

// Set the model ID, e.g., Mistral Large.
var modelId = "mistral.mistral-large-2402-v1:0";
```

```
// Define the user message.
var userMessage = "Describe the purpose of a 'hello world' program in one line.";

// Create a request with the model ID, the user message, and an inference
 configuration.
var request = new ConverseStreamRequest
{
    ModelId = modelId,
    Messages = new List<Message>
    {
        new Message
        {
            Role = ConversationRole.User,
            Content = new List<ContentBlock> { new ContentBlock { Text =
 userMessage } }
        }
    },
    InferenceConfig = new InferenceConfiguration()
    {
        MaxTokens = 512,
        Temperature = 0.5F,
        TopP = 0.9F
    }
};

try
{
    // Send the request to the Bedrock Runtime and wait for the result.
    var response = await client.ConverseStreamAsync(request);

    // Extract and print the streamed response text in real-time.
    foreach (var chunk in response.Stream.AsEnumerable())
    {
        if (chunk is ContentBlockDeltaEvent)
        {
            Console.Write((chunk as ContentBlockDeltaEvent).Delta.Text);
        }
    }
}
catch (AmazonBedrockRuntimeException e)
{
    Console.WriteLine($"ERROR: Can't invoke '{modelId}'. Reason: {e.Message}");
    throw;
```

```
}
```

- Para obter detalhes da API, consulte ConverseStreama Referência AWS SDK para .NET da API.

CloudFormation exemplos usando SDK para .NET (v4)

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET (v4) com CloudFormation.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- Conceitos básicos

Conceitos básicos

Olá CloudFormation

O exemplo de código a seguir mostra como começar a usar o CloudFormation.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
using Amazon.CloudFormation;
using Amazon.CloudFormation.Model;
using Amazon.Runtime;

namespace CloudFormationActions;

public static class HelloCloudFormation
```

```
{
    public static IAmazonCloudFormation _amazonCloudFormation = null!;

    static async Task Main(string[] args)
    {
        // Create the CloudFormation client
        _amazonCloudFormation = new AmazonCloudFormationClient();
        Console.WriteLine($"\nIn Region:
 {_amazonCloudFormation.Config.RegionEndpoint}");

        // List the resources for each stack
        await ListResources();
    }

    /// <summary>
    /// Method to list stack resources and other information.
    /// </summary>
    /// <returns>True if successful.</returns>
    public static async Task<bool> ListResources()
    {
        try
        {
            Console.WriteLine("Getting CloudFormation stack information...");

            // Get all stacks using the stack paginator.
            var paginatorForDescribeStacks =
                _amazonCloudFormation.Paginators.DescribeStacks(
                    new DescribeStacksRequest());
            if (paginatorForDescribeStacks.Stacks != null)
            {
                await foreach (Stack stack in paginatorForDescribeStacks.Stacks)
                {
                    // Basic information for each stack
                    Console.WriteLine(
                        "\n----------------------------------------------");
                    Console.WriteLine($"\nStack: {stack.StackName}");
                    Console.WriteLine($"  Status: {stack.StackStatus.Value}");
                    Console.WriteLine($"  Created: {stack.CreationTime}");

                    // The tags of each stack (etc.)
                    if (stack.Tags != null && stack.Tags.Count > 0)
                    {
                        Console.WriteLine("  Tags:");
                        foreach (Tag tag in stack.Tags)
```

```
                        Console.WriteLine($"    {tag.Key}, {tag.Value}");
                }

                // The resources of each stack
                DescribeStackResourcesResponse responseDescribeResources =
                    await _amazonCloudFormation.DescribeStackResourcesAsync(
                        new DescribeStackResourcesRequest
                        {
                            StackName = stack.StackName
                        });
                if (responseDescribeResources.StackResources != null &&
responseDescribeResources.StackResources.Count > 0)
                {
                    Console.WriteLine("  Resources:");
                    foreach (StackResource resource in responseDescribeResources
                                .StackResources)
                        Console.WriteLine(
                            $"    {resource.LogicalResourceId}:
{resource.ResourceStatus}");
                }
            }
        }

        Console.WriteLine("\n---------------------------------------------");
        return true;
    }
    catch (AmazonCloudFormationException ex)
    {
        Console.WriteLine("Unable to get stack information:\n" + ex.Message);
        return false;
    }
    catch (AmazonServiceException ex)
    {
        if (ex.Message.Contains("Unable to get IAM security credentials"))
        {
            Console.WriteLine(ex.Message);
            Console.WriteLine("If you are usnig SSO, be sure to install" +
                              " the AWSSDK.SSO and AWSSDK.SSOOIDC packages.");
        }
        else
        {
            Console.WriteLine(ex.Message);
            Console.WriteLine(ex.StackTrace);
        }
```

```
            return false;
        }
        catch (ArgumentNullException ex)
        {
            if (ex.Message.Contains("Options property cannot be empty: ClientName"))
            {
                Console.WriteLine(ex.Message);
                Console.WriteLine("If you are using SSO, have you logged in?");
            }
            else
            {
                Console.WriteLine(ex.Message);
                Console.WriteLine(ex.StackTrace);
            }

            return false;
        }
    }
```

- Para obter detalhes da API, consulte DescribeStackResourcesa Referência AWS SDK para .NET da API.

CloudWatch exemplos usando SDK para .NET (v4)

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET (v4) com CloudWatch.

As noções básicas são exemplos de código que mostram como realizar as operações essenciais em um serviço.

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar perfis de serviço individuais, você pode ver as ações no contexto em seus cenários relacionados.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- Conceitos básicos

- Conceitos básicos
- Ações

Conceitos básicos

Olá CloudWatch

O exemplo de código a seguir mostra como começar a usar o CloudWatch.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
using Amazon.CloudWatch;
using Amazon.CloudWatch.Model;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

namespace CloudWatchActions;

public static class HelloCloudWatch
{
    static async Task Main(string[] args)
    {
        // Use the AWS .NET Core Setup package to set up dependency injection for
 the Amazon CloudWatch service.
        // Use your AWS profile name, or leave it blank to use the default profile.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonCloudWatch>()
            ).Build();

        // Now the client is available for injection.
        var cloudWatchClient =
 host.Services.GetRequiredService<IAmazonCloudWatch>();

        // You can use await and any of the async methods to get a response.
```

```
        var metricNamespace = "AWS/Billing";
        var response = await cloudWatchClient.ListMetricsAsync(new
 ListMetricsRequest
        {
            Namespace = metricNamespace
        });
        Console.WriteLine($"Hello Amazon CloudWatch! Following are some metrics
 available in the {metricNamespace} namespace:");
        Console.WriteLine();
        if (response.Metrics != null)
        {
            foreach (var metric in response.Metrics.Take(5))
            {
                Console.WriteLine($"\tMetric: {metric.MetricName}");
                Console.WriteLine($"\tNamespace: {metric.Namespace}");
                Console.WriteLine(
                    $"\tDimensions: {string.Join(", ", metric.Dimensions.Select(m =>
 $"{m.Name}:{m.Value}"))}");
                Console.WriteLine();
            }
        }
    }
}
```

- Para obter detalhes da API, consulte ListMetricsa Referência AWS SDK para .NET da API.

Conceitos básicos

Conheça os conceitos básicos

O exemplo de código a seguir mostra como:

- Listar CloudWatch namespaces e métricas.
- Obter estatísticas para uma métrica e para faturamento estimado.
- Criar e atualizar um painel.
- Criar e adicionar dados a uma métrica.
- Criar e acionar um alarme e, em seguida, visualizar o histórico de alarmes.
- Criar um detector de anomalias.
- Obter uma imagem de métrica e, em seguida, limpar os recursos.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Execute um cenário interativo em um prompt de comando.

```
public class CloudWatchScenario
{
    /*
    Before running this .NET code example, set up your development environment,
 including your credentials.

    To enable billing metrics and statistics for this example, make sure billing
 alerts are enabled for your account:
    https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/
monitor_estimated_charges_with_cloudwatch.html#turning_on_billing_metrics

    This .NET example performs the following tasks:
        1. List and select a CloudWatch namespace.
        2. List and select a CloudWatch metric.
        3. Get statistics for a CloudWatch metric.
        4. Get estimated billing statistics for the last week.
        5. Create a new CloudWatch dashboard with two metrics.
        6. List current CloudWatch dashboards.
        7. Create a CloudWatch custom metric and add metric data.
        8. Add the custom metric to the dashboard.
        9. Create a CloudWatch alarm for the custom metric.
       10. Describe current CloudWatch alarms.
       11. Get recent data for the custom metric.
       12. Add data to the custom metric to trigger the alarm.
       13. Wait for an alarm state.
       14. Get history for the CloudWatch alarm.
       15. Add an anomaly detector.
       16. Describe current anomaly detectors.
       17. Get and display a metric image.
       18. Clean up resources.
    */

    private static ILogger logger = null!;
```

```
    private static CloudWatchWrapper _cloudWatchWrapper = null!;
    private static IConfiguration _configuration = null!;
    private static readonly List<string> _statTypes = new List<string>
{ "SampleCount", "Average", "Sum", "Minimum", "Maximum" };
    private static SingleMetricAnomalyDetector? anomalyDetector = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for the Amazon service.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
            .ConfigureServices((_, services) =>
            services.AddAWSService<IAmazonCloudWatch>()
            .AddTransient<CloudWatchWrapper>()
        )
        .Build();

        _configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("settings.json") // Load settings from .json file.
            .AddJsonFile("settings.local.json",
                true) // Optionally, load local settings.
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<CloudWatchScenario>();

        _cloudWatchWrapper = host.Services.GetRequiredService<CloudWatchWrapper>();

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Welcome to the Amazon CloudWatch example scenario.");
        Console.WriteLine(new string('-', 80));

        try
        {
            var selectedNamespace = await SelectNamespace();
            var selectedMetric = await SelectMetric(selectedNamespace);
            await GetAndDisplayMetricStatistics(selectedNamespace, selectedMetric);
            await GetAndDisplayEstimatedBilling();
            await CreateDashboardWithMetrics();
```

```
            await ListDashboards();
            await CreateNewCustomMetric();
            await AddMetricToDashboard();
            await CreateMetricAlarm();
            await DescribeAlarms();
            await GetCustomMetricData();
            await AddMetricDataForAlarm();
            await CheckForMetricAlarm();
            await GetAlarmHistory();
            anomalyDetector = await AddAnomalyDetector();
            await DescribeAnomalyDetectors();
            await GetAndOpenMetricImage();
            await CleanupResources();
        }
        catch (Exception ex)
        {
            logger.LogError(ex, "There was a problem executing the scenario.");
            await CleanupResources();
        }

    }

    /// <summary>
    /// Select a namespace.
    /// </summary>
    /// <returns>The selected namespace.</returns>
    private static async Task<string> SelectNamespace()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"1. Select a CloudWatch Namespace from a list of
Namespaces.");
        var metrics = await _cloudWatchWrapper.ListMetrics();
        // Get a distinct list of namespaces.
        var namespaces = metrics.Select(m => m.Namespace).Distinct().ToList();
        for (int i = 0; i < namespaces.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. {namespaces[i]}");
        }

        var namespaceChoiceNumber = 0;
        while (namespaceChoiceNumber < 1 || namespaceChoiceNumber >
namespaces.Count)
        {
            Console.WriteLine(
```

```
                "Select a namespace by entering a number from the preceding list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out namespaceChoiceNumber);
        }

        var selectedNamespace = namespaces[namespaceChoiceNumber - 1];

        Console.WriteLine(new string('-', 80));

        return selectedNamespace;
    }

    /// <summary>
    /// Select a metric from a namespace.
    /// </summary>
    /// <param name="metricNamespace">The namespace for metrics.</param>
    /// <returns>The metric name.</returns>
    private static async Task<Metric> SelectMetric(string metricNamespace)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"2. Select a CloudWatch metric from a namespace.");

        var namespaceMetrics = await
_cloudWatchWrapper.ListMetrics(metricNamespace);

        for (int i = 0; i < namespaceMetrics.Count && i < 15; i++)
        {
            var dimensionsWithValues = namespaceMetrics[i].Dimensions
                .Where(d => !string.Equals("None", d.Value));
            Console.WriteLine($"\t{i + 1}. {namespaceMetrics[i].MetricName} " +
                              $"{string.Join(", :", dimensionsWithValues.Select(d =>
d.Value))}");
        }

        var metricChoiceNumber = 0;
        while (metricChoiceNumber < 1 || metricChoiceNumber >
namespaceMetrics.Count)
        {
            Console.WriteLine(
                "Select a metric by entering a number from the preceding list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out metricChoiceNumber);
        }
```

```
        var selectedMetric = namespaceMetrics[metricChoiceNumber - 1];

        Console.WriteLine(new string('-', 80));

        return selectedMetric;
    }

    /// <summary>
    /// Get and display metric statistics for a specific metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace for metrics.</param>
    /// <param name="metric">The CloudWatch metric.</param>
    /// <returns>Async task.</returns>
    private static async Task GetAndDisplayMetricStatistics(string metricNamespace,
Metric metric)
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"3. Get CloudWatch metric statistics for the last day.");

        for (int i = 0; i < _statTypes.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. {_statTypes[i]}");
        }

        var statisticChoiceNumber = 0;
        while (statisticChoiceNumber < 1 || statisticChoiceNumber >
_statTypes.Count)
        {
            Console.WriteLine(
                "Select a metric statistic by entering a number from the preceding
list:");
            var choice = Console.ReadLine();
            Int32.TryParse(choice, out statisticChoiceNumber);
        }

        var selectedStatistic = _statTypes[statisticChoiceNumber - 1];
        var statisticsList = new List<string> { selectedStatistic };

        var metricStatistics = await
_cloudWatchWrapper.GetMetricStatistics(metricNamespace, metric.MetricName,
statisticsList, metric.Dimensions, 1, 60);

        if (!metricStatistics.Any())
        {
```

```
            Console.WriteLine($"No {selectedStatistic} statistics found for {metric}
in namespace {metricNamespace}.");
        }

        metricStatistics = metricStatistics.OrderBy(s => s.Timestamp).ToList();
        for (int i = 0; i < metricStatistics.Count && i < 10; i++)
        {
            var metricStat = metricStatistics[i];
            var statValue =
metricStat.GetType().GetProperty(selectedStatistic)!.GetValue(metricStat, null);
            Console.WriteLine($"\t{i + 1}. Timestamp
{metricStatistics[i].Timestamp:G} {selectedStatistic}: {statValue}");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Get and display estimated billing statistics.
    /// </summary>
    /// <param name="metricNamespace">The namespace for metrics.</param>
    /// <param name="metric">The CloudWatch metric.</param>
    /// <returns>Async task.</returns>
    private static async Task GetAndDisplayEstimatedBilling()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"4. Get CloudWatch estimated billing for the last
week.");

        var billingStatistics = await SetupBillingStatistics();

        for (int i = 0; i < billingStatistics.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. Timestamp
{billingStatistics[i].Timestamp:G} : {billingStatistics[i].Maximum}");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Get billing statistics using a call to a wrapper class.
    /// </summary>
    /// <returns>A collection of billing statistics.</returns>
```

```
    private static async Task<List<Datapoint>> SetupBillingStatistics()
    {
        // Make a request for EstimatedCharges with a period of one day for the past
seven days.
        var billingStatistics = await _cloudWatchWrapper.GetMetricStatistics(
            "AWS/Billing",
            "EstimatedCharges",
            new List<string>() { "Maximum" },
            new List<Dimension>() { new Dimension { Name = "Currency", Value =
"USD" } },
            7,
            86400);

        billingStatistics = billingStatistics.OrderBy(n => n.Timestamp).ToList();

        return billingStatistics;
    }

    /// <summary>
    /// Create a dashboard with metrics.
    /// </summary>
    /// <param name="metricNamespace">The namespace for metrics.</param>
    /// <param name="metric">The CloudWatch metric.</param>
    /// <returns>Async task.</returns>
    private static async Task CreateDashboardWithMetrics()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"5. Create a new CloudWatch dashboard with metrics.");
        var dashboardName = _configuration["dashboardName"];
        var newDashboard = new DashboardModel();
        _configuration.GetSection("dashboardExampleBody").Bind(newDashboard);
        var newDashboardString = JsonSerializer.Serialize(
            newDashboard,
            new JsonSerializerOptions
            {
                DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingNull
            });
        var validationMessages =
            await _cloudWatchWrapper.PutDashboard(dashboardName,
newDashboardString);

        Console.WriteLine(validationMessages.Any() ? $"\tValidation messages:" :
null);
        for (int i = 0; i < validationMessages.Count; i++)
```

```
        {
            Console.WriteLine($"\t{i + 1}. {validationMessages[i].Message}");
        }
        Console.WriteLine($"\tDashboard {dashboardName} was created.");
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// List dashboards.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task ListDashboards()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"6. List the CloudWatch dashboards in the current
account.");

        var dashboards = await _cloudWatchWrapper.ListDashboards();

        for (int i = 0; i < dashboards.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. {dashboards[i].DashboardName}");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Create and add data for a new custom metric.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task CreateNewCustomMetric()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"7. Create and add data for a new custom metric.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];

        var customData = await PutRandomMetricData(customMetricName,
customMetricNamespace);

        var valuesString = string.Join(',', customData.Select(d => d.Value));
```

```
        Console.WriteLine($"\tAdded metric values for for metric {customMetricName}:
\n\t{valuesString}");

        Console.WriteLine(new string('-', 80));
    }


    /// <summary>
    /// Add some metric data using a call to a wrapper class.
    /// </summary>
    /// <param name="customMetricName">The metric name.</param>
    /// <param name="customMetricNamespace">The metric namespace.</param>
    /// <returns></returns>
    private static async Task<List<MetricDatum>> PutRandomMetricData(string
customMetricName,
        string customMetricNamespace)
    {
        List<MetricDatum> customData = new List<MetricDatum>();
        Random rnd = new Random();

        // Add 10 random values up to 100, starting with a timestamp 15 minutes in
the past.
        var utcNowMinus15 = DateTime.UtcNow.AddMinutes(-15);
        for (int i = 0; i < 10; i++)
        {
            var metricValue = rnd.Next(0, 100);
            customData.Add(
                new MetricDatum
                {
                    MetricName = customMetricName,
                    Value = metricValue,
                    TimestampUtc = utcNowMinus15.AddMinutes(i)
                }
            );
        }

        await _cloudWatchWrapper.PutMetricData(customMetricNamespace, customData);
        return customData;
    }

    /// <summary>
    /// Add the custom metric to the dashboard.
    /// </summary>
    /// <returns>Async task.</returns>
```

```
    private static async Task AddMetricToDashboard()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"8. Add the new custom metric to the dashboard.");

        var dashboardName = _configuration["dashboardName"];

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];

        var validationMessages = await SetupDashboard(customMetricNamespace,
customMetricName, dashboardName);

        Console.WriteLine(validationMessages.Any() ? $"\tValidation messages:" :
null);
        for (int i = 0; i < validationMessages.Count; i++)
        {
            Console.WriteLine($"\t{i + 1}. {validationMessages[i].Message}");
        }
        Console.WriteLine($"\tDashboard {dashboardName} updated with metric
{customMetricName}.");
        Console.WriteLine(new string('-', 80));
    }


    /// <summary>
    /// Set up a dashboard using a call to the wrapper class.
    /// </summary>
    /// <param name="customMetricNamespace">The metric namespace.</param>
    /// <param name="customMetricName">The metric name.</param>
    /// <param name="dashboardName">The name of the dashboard.</param>
    /// <returns>A list of validation messages.</returns>
    private static async Task<List<DashboardValidationMessage>> SetupDashboard(
        string customMetricNamespace, string customMetricName, string dashboardName)
    {
        // Get the dashboard model from configuration.
        var newDashboard = new DashboardModel();
        _configuration.GetSection("dashboardExampleBody").Bind(newDashboard);

        // Add a new metric to the dashboard.
        newDashboard.Widgets.Add(new Widget
        {
            Height = 8,
            Width = 8,
```

```
            Y = 8,
            X = 0,
            Type = "metric",
            Properties = new Properties
            {
                Metrics = new List<List<object>>
                    { new() { customMetricNamespace, customMetricName } },
                View = "timeSeries",
                Region = "us-east-1",
                Stat = "Sum",
                Period = 86400,
                YAxis = new YAxis { Left = new Left { Min = 0, Max = 100 } },
                Title = "Custom Metric Widget",
                LiveData = true,
                Sparkline = true,
                Trend = true,
                Stacked = false,
                SetPeriodToTimeRange = false
            }
        });

        var newDashboardString = JsonSerializer.Serialize(newDashboard,
            new JsonSerializerOptions
            { DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingNull });
        var validationMessages =
            await _cloudWatchWrapper.PutDashboard(dashboardName,
newDashboardString);

        return validationMessages;
    }

    /// <summary>
    /// Create a CloudWatch alarm for the new metric.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task CreateMetricAlarm()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"9. Create a CloudWatch alarm for the new metric.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];

        var alarmName = _configuration["exampleAlarmName"];
```

```
        var accountId = _configuration["accountId"];
        var region = _configuration["region"];
        var emailTopic = _configuration["emailTopic"];
        var alarmActions = new List<string>();

        if (GetYesNoResponse(
                $"\tAdd an email action for topic {emailTopic} to alarm {alarmName}?
(y/n)"))
        {
            _cloudWatchWrapper.AddEmailAlarmAction(accountId, region, emailTopic,
alarmActions);
        }

        await _cloudWatchWrapper.PutMetricEmailAlarm(
            "Example metric alarm",
            alarmName,
            ComparisonOperator.GreaterThanOrEqualToThreshold,
            customMetricName,
            customMetricNamespace,
            100,
            alarmActions);

        Console.WriteLine($"\tAlarm {alarmName} added for metric
{customMetricName}.");
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Describe Alarms.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task DescribeAlarms()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"10. Describe CloudWatch alarms in the current
account.");

        var alarms = await _cloudWatchWrapper.DescribeAlarms();
        alarms = alarms.OrderByDescending(a => a.StateUpdatedTimestamp).ToList();

        for (int i = 0; i < alarms.Count && i < 10; i++)
        {
            var alarm = alarms[i];
            Console.WriteLine($"\t{i + 1}. {alarm.AlarmName}");
```

```
            Console.WriteLine($"\tState: {alarm.StateValue} for {alarm.MetricName}
{alarm.ComparisonOperator} {alarm.Threshold}");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Get the recent data for the metric.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task GetCustomMetricData()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"11. Get current data for new custom metric.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];
        var accountId = _configuration["accountId"];

        var query = new List<MetricDataQuery>
        {
            new MetricDataQuery
            {
                AccountId = accountId,
                Id = "m1",
                Label = "Custom Metric Data",
                MetricStat = new MetricStat
                {
                    Metric = new Metric
                    {
                        MetricName = customMetricName,
                        Namespace = customMetricNamespace,
                    },
                    Period = 1,
                    Stat = "Maximum"
                }
            }
        };

        var metricData = await _cloudWatchWrapper.GetMetricData(
            20,
            true,
            DateTime.UtcNow.AddMinutes(1),
```

```
            20,
            query);

        for (int i = 0; i < metricData.Count; i++)
        {
            if (metricData[i].Values != null)
            {
                for (int j = 0; j < metricData[i].Values.Count; j++)
                {
                    Console.WriteLine(
                        $"\tTimestamp {metricData[i].Timestamps[j]:G} Value:
{metricData[i].Values[j]}");
                }
            }
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Add metric data to trigger an alarm.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task AddMetricDataForAlarm()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"12. Add metric data to the custom metric to trigger an
alarm.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];
        var nowUtc = DateTime.UtcNow;
        List<MetricDatum> customData = new List<MetricDatum>
        {
            new MetricDatum
            {
                MetricName = customMetricName,
                Value = 101,
                TimestampUtc = nowUtc.AddMinutes(-2)
            },
            new MetricDatum
            {
                MetricName = customMetricName,
                Value = 101,
```

```
                TimestampUtc = nowUtc.AddMinutes(-1)
            },
            new MetricDatum
            {
                MetricName = customMetricName,
                Value = 101,
                TimestampUtc = nowUtc
            }
        };
        var valuesString = string.Join(',', customData.Select(d => d.Value));
        Console.WriteLine($"\tAdded metric values for for metric {customMetricName}:
\n\t{valuesString}");
        await _cloudWatchWrapper.PutMetricData(customMetricNamespace, customData);

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Check for a metric alarm using the DescribeAlarmsForMetric action.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task CheckForMetricAlarm()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"13. Checking for an alarm state.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];
        var hasAlarm = false;
        var retries = 10;
        while (!hasAlarm && retries > 0)
        {
            var alarms = await
_cloudWatchWrapper.DescribeAlarmsForMetric(customMetricNamespace,
customMetricName);
            hasAlarm = alarms.Any(a => a.StateValue == StateValue.ALARM);
            retries--;
            Thread.Sleep(20000);
        }

        Console.WriteLine(hasAlarm
            ? $"\tAlarm state found for {customMetricName}."
            : $"\tNo Alarm state found for {customMetricName} after 10 retries.");
```

```
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Get history for an alarm.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task GetAlarmHistory()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"14. Get alarm history.");

        var exampleAlarmName = _configuration["exampleAlarmName"];

        var alarmHistory = await
_cloudWatchWrapper.DescribeAlarmHistory(exampleAlarmName, 2);

        for (int i = 0; i < alarmHistory.Count; i++)
        {
            var history = alarmHistory[i];
            Console.WriteLine($"\t{i + 1}. {history.HistorySummary}, time
{history.Timestamp:g}");
        }
        if (!alarmHistory.Any())
        {
            Console.WriteLine($"\tNo alarm history data found for
{exampleAlarmName}.");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Add an anomaly detector.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task<SingleMetricAnomalyDetector> AddAnomalyDetector()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"15. Add an anomaly detector.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];
```

```
        var detector = new SingleMetricAnomalyDetector
        {
            MetricName = customMetricName,
            Namespace = customMetricNamespace,
            Stat = "Maximum"
        };
        await _cloudWatchWrapper.PutAnomalyDetector(detector);
        Console.WriteLine($"\tAdded anomaly detector for metric
{customMetricName}.");

        Console.WriteLine(new string('-', 80));
        return detector;
    }

    /// <summary>
    /// Describe anomaly detectors.
    /// </summary>
    /// <returns>Async task.</returns>
    private static async Task DescribeAnomalyDetectors()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"16. Describe anomaly detectors in the current
account.");

        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];

        var detectors = await
_cloudWatchWrapper.DescribeAnomalyDetectors(customMetricNamespace,
customMetricName);

        for (int i = 0; i < detectors.Count; i++)
        {
            var detector = detectors[i];
            Console.WriteLine($"\t{i + 1}.
{detector.SingleMetricAnomalyDetector.MetricName}, state {detector.StateValue}");
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Fetch and open a metrics image for a CloudWatch metric and namespace.
    /// </summary>
```

```
    /// <returns>Async task.</returns>
    private static async Task GetAndOpenMetricImage()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("17. Get a metric image from CloudWatch.");

        Console.WriteLine($"\tGetting Image data for custom metric.");
        var customMetricNamespace = _configuration["customMetricNamespace"];
        var customMetricName = _configuration["customMetricName"];

        var memoryStream = await
_cloudWatchWrapper.GetTimeSeriesMetricImage(customMetricNamespace,
customMetricName, "Maximum", 10);
        var file = _cloudWatchWrapper.SaveMetricImage(memoryStream, "MetricImages");

        ProcessStartInfo info = new ProcessStartInfo();

        Console.WriteLine($"\tFile saved as {Path.GetFileName(file)}.");
        Console.WriteLine($"\tPress enter to open the image.");
        Console.ReadLine();
        info.FileName = Path.Combine("ms-photos://", file);
        info.UseShellExecute = true;
        info.CreateNoWindow = true;
        info.Verb = string.Empty;

        Process.Start(info);

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Clean up created resources.
    /// </summary>
    /// <param name="metricNamespace">The namespace for metrics.</param>
    /// <param name="metric">The CloudWatch metric.</param>
    /// <returns>Async task.</returns>
    private static async Task CleanupResources()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine($"18. Clean up resources.");

        var dashboardName = _configuration["dashboardName"];
        if (GetYesNoResponse($"\tDelete dashboard {dashboardName}? (y/n)"))
        {
```

```
            Console.WriteLine($"\tDeleting dashboard.");
            var dashboardList = new List<string> { dashboardName };
            await _cloudWatchWrapper.DeleteDashboards(dashboardList);
        }

        var alarmName = _configuration["exampleAlarmName"];
        if (GetYesNoResponse($"\tDelete alarm {alarmName}? (y/n)"))
        {
            Console.WriteLine($"\tCleaning up alarms.");
            var alarms = new List<string> { alarmName };
            await _cloudWatchWrapper.DeleteAlarms(alarms);
        }

        if (GetYesNoResponse($"\tDelete anomaly detector? (y/n)") &&
 anomalyDetector != null)
        {
            Console.WriteLine($"\tCleaning up anomaly detector.");

            await _cloudWatchWrapper.DeleteAnomalyDetector(
                anomalyDetector);
        }

        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Get a yes or no response from the user.
    /// </summary>
    /// <param name="question">The question string to print on the console.</param>
    /// <returns>True if the user responds with a yes.</returns>
    private static bool GetYesNoResponse(string question)
    {
        Console.WriteLine(question);
        var ynResponse = Console.ReadLine();
        var response = ynResponse != null &&
                       ynResponse.Equals("y",
                           StringComparison.InvariantCultureIgnoreCase);
        return response;
    }
}
```

Métodos de embalagem usados pelo cenário para CloudWatch ações.

```
/// <summary>
/// Wrapper class for Amazon CloudWatch methods.
/// </summary>
public class CloudWatchWrapper
{
    private readonly IAmazonCloudWatch _amazonCloudWatch;
    private readonly ILogger<CloudWatchWrapper> _logger;

    /// <summary>
    /// Constructor for the CloudWatch wrapper.
    /// </summary>
    /// <param name="amazonCloudWatch">The injected CloudWatch client.</param>
    /// <param name="logger">The injected logger for the wrapper.</param>
    public CloudWatchWrapper(IAmazonCloudWatch amazonCloudWatch,
 ILogger<CloudWatchWrapper> logger)

    {
        _logger = logger;
        _amazonCloudWatch = amazonCloudWatch;
    }

    /// <summary>
    /// List metrics available, optionally within a namespace.
    /// </summary>
    /// <param name="metricNamespace">Optional CloudWatch namespace to use when
 listing metrics.</param>
    /// <param name="filter">Optional dimension filter.</param>
    /// <param name="metricName">Optional metric name filter.</param>
    /// <returns>The list of metrics.</returns>
    public async Task<List<Metric>> ListMetrics(string? metricNamespace = null,
 DimensionFilter? filter = null, string? metricName = null)
    {
        var results = new List<Metric>();
        var paginateMetrics = _amazonCloudWatch.Paginators.ListMetrics(
            new ListMetricsRequest
            {
                Namespace = metricNamespace,
                Dimensions = filter != null ? new List<DimensionFilter> { filter } :
 null,
                MetricName = metricName
            });
        // Get the entire list using the paginator.
        await foreach (var metric in paginateMetrics.Metrics)
```

```
        {
            results.Add(metric);
        }

        return results;
    }

    /// <summary>
    /// Wrapper to get statistics for a specific CloudWatch metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricName">The name of the metric.</param>
    /// <param name="statistics">The list of statistics to include.</param>
    /// <param name="dimensions">The list of dimensions to include.</param>
    /// <param name="days">The number of days in the past to include.</param>
    /// <param name="period">The period for the data.</param>
    /// <returns>A list of DataPoint objects for the statistics.</returns>
    public async Task<List<Datapoint>> GetMetricStatistics(string metricNamespace,
        string metricName, List<string> statistics, List<Dimension> dimensions, int
 days, int period)
    {
        var metricStatistics = await _amazonCloudWatch.GetMetricStatisticsAsync(
            new GetMetricStatisticsRequest()
            {
                Namespace = metricNamespace,
                MetricName = metricName,
                Dimensions = dimensions,
                Statistics = statistics,
                StartTimeUtc = DateTime.UtcNow.AddDays(-days),
                EndTimeUtc = DateTime.UtcNow,
                Period = period
            });

        return metricStatistics.Datapoints ?? new List<Datapoint>();
    }

    /// <summary>
    /// Wrapper to create or add to a dashboard with metrics.
    /// </summary>
    /// <param name="dashboardName">The name for the dashboard.</param>
    /// <param name="dashboardBody">The metric data in JSON for the dashboard.</
param>
    /// <returns>A list of validation messages for the dashboard.</returns>
```

```
   public async Task<List<DashboardValidationMessage>> PutDashboard(string
dashboardName,
       string dashboardBody)
   {
       // Updating a dashboard replaces all contents.
       // Best practice is to include a text widget indicating this dashboard was
created programmatically.
       var dashboardResponse = await _amazonCloudWatch.PutDashboardAsync(
           new PutDashboardRequest()
           {
               DashboardName = dashboardName,
               DashboardBody = dashboardBody
           });

       return dashboardResponse.DashboardValidationMessages ?? new
List<DashboardValidationMessage>();
   }


   /// <summary>
   /// Get information on a dashboard.
   /// </summary>
   /// <param name="dashboardName">The name of the dashboard.</param>
   /// <returns>A JSON object with dashboard information.</returns>
   public async Task<string> GetDashboard(string dashboardName)
   {
       var dashboardResponse = await _amazonCloudWatch.GetDashboardAsync(
           new GetDashboardRequest()
           {
               DashboardName = dashboardName
           });

       return dashboardResponse.DashboardBody;
   }


   /// <summary>
   /// Get a list of dashboards.
   /// </summary>
   /// <returns>A list of DashboardEntry objects.</returns>
   public async Task<List<DashboardEntry>> ListDashboards()
   {
       var results = new List<DashboardEntry>();
       var paginateDashboards = _amazonCloudWatch.Paginators.ListDashboards(
```

```
            new ListDashboardsRequest());
        // Get the entire list using the paginator.
        await foreach (var data in paginateDashboards.DashboardEntries)
        {
            results.Add(data);
        }

        return results;
    }

    /// <summary>
    /// Wrapper to add metric data to a CloudWatch metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricData">A data object for the metric data.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> PutMetricData(string metricNamespace,
        List<MetricDatum> metricData)
    {
        var putDataResponse = await _amazonCloudWatch.PutMetricDataAsync(
            new PutMetricDataRequest()
            {
                MetricData = metricData,
                Namespace = metricNamespace,
            });

        return putDataResponse.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Get an image for a metric graphed over time.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metric">The name of the metric.</param>
    /// <param name="stat">The name of the stat to chart.</param>
    /// <param name="period">The period to use for the chart.</param>
    /// <returns>A memory stream for the chart image.</returns>
    public async Task<MemoryStream> GetTimeSeriesMetricImage(string metricNamespace,
string metric, string stat, int period)
    {
        var metricImageWidget = new
        {
            title = "Example Metric Graph",
            view = "timeSeries",
```

```
            stacked = false,
            period = period,
            width = 1400,
            height = 600,
            metrics = new List<List<object>>
                { new() { metricNamespace, metric, new { stat } } }
        };

        var metricImageWidgetString = JsonSerializer.Serialize(metricImageWidget);
        var imageResponse = await _amazonCloudWatch.GetMetricWidgetImageAsync(
            new GetMetricWidgetImageRequest()
            {
                MetricWidget = metricImageWidgetString
            });

        return imageResponse.MetricWidgetImage;
    }

    /// <summary>
    /// Save a metric image to a file.
    /// </summary>
    /// <param name="memoryStream">The MemoryStream for the metric image.</param>
    /// <param name="metricName">The name of the metric.</param>
    /// <returns>The path to the file.</returns>
    public string SaveMetricImage(MemoryStream memoryStream, string metricName)
    {
        var metricFileName = $"{metricName}_{DateTime.Now.Ticks}.png";
        using var sr = new StreamReader(memoryStream);
        // Writes the memory stream to a file.
        File.WriteAllBytes(metricFileName, memoryStream.ToArray());
        var filePath = Path.Join(AppDomain.CurrentDomain.BaseDirectory,
            metricFileName);
        return filePath;
    }

    /// <summary>
    /// Get data for CloudWatch metrics.
    /// </summary>
    /// <param name="minutesOfData">The number of minutes of data to include.</
param>
    /// <param name="useDescendingTime">True to return the data descending by
 time.</param>
    /// <param name="endDateUtc">The end date for the data, in UTC.</param>
    /// <param name="maxDataPoints">The maximum data points to include.</param>
```

```
    /// <param name="dataQueries">Optional data queries to include.</param>
    /// <returns>A list of the requested metric data.</returns>
    public async Task<List<MetricDataResult>> GetMetricData(int minutesOfData, bool
 useDescendingTime, DateTime? endDateUtc = null,
        int maxDataPoints = 0, List<MetricDataQuery>? dataQueries = null)
    {
        var metricData = new List<MetricDataResult>();
        // If no end time is provided, use the current time for the end time.
        endDateUtc ??= DateTime.UtcNow;
        var timeZoneOffset =
 TimeZoneInfo.Local.GetUtcOffset(endDateUtc.Value.ToLocalTime());
        var startTimeUtc = endDateUtc.Value.AddMinutes(-minutesOfData);
        // The timezone string should be in the format +0000, so use the timezone
 offset to format it correctly.
        var timeZoneString = $"{timeZoneOffset.Hours:D2}
{timeZoneOffset.Minutes:D2}";
        // Add the plus sign for positive offsets.
        timeZoneString = timeZoneString.StartsWith('-') ? timeZoneString : "+" +
 timeZoneString;
        var paginatedMetricData = _amazonCloudWatch.Paginators.GetMetricData(
            new GetMetricDataRequest()
            {
                StartTimeUtc = startTimeUtc,
                EndTimeUtc = endDateUtc.Value,
                LabelOptions = new LabelOptions { Timezone = timeZoneString },
                ScanBy = useDescendingTime ? ScanBy.TimestampDescending :
 ScanBy.TimestampAscending,
                MaxDatapoints = maxDataPoints,
                MetricDataQueries = dataQueries,
            });

        if (paginatedMetricData.MetricDataResults != null)
        {
            await foreach (var data in paginatedMetricData.MetricDataResults)
            {
                metricData.Add(data);
            }
        }

        return metricData;
    }

    /// <summary>
    /// Add a metric alarm to send an email when the metric passes a threshold.
```

```
   /// </summary>
   /// <param name="alarmDescription">A description of the alarm.</param>
   /// <param name="alarmName">The name for the alarm.</param>
   /// <param name="comparison">The type of comparison to use.</param>
   /// <param name="metricName">The name of the metric for the alarm.</param>
   /// <param name="metricNamespace">The namespace of the metric.</param>
   /// <param name="threshold">The threshold value for the alarm.</param>
   /// <param name="alarmActions">Optional actions to execute when in an alarm
state.</param>
   /// <returns>True if successful.</returns>
   public async Task<bool> PutMetricEmailAlarm(string alarmDescription, string
alarmName, ComparisonOperator comparison,
       string metricName, string metricNamespace, double threshold, List<string>
alarmActions = null!)
   {
       try
       {
           var putEmailAlarmResponse = await _amazonCloudWatch.PutMetricAlarmAsync(
               new PutMetricAlarmRequest()
               {
                   AlarmActions = alarmActions,
                   AlarmDescription = alarmDescription,
                   AlarmName = alarmName,
                   ComparisonOperator = comparison,
                   Threshold = threshold,
                   Namespace = metricNamespace,
                   MetricName = metricName,
                   EvaluationPeriods = 1,
                   Period = 10,
                   Statistic = new Statistic("Maximum"),
                   DatapointsToAlarm = 1,
                   TreatMissingData = "ignore"
               });
           return putEmailAlarmResponse.HttpStatusCode == HttpStatusCode.OK;
       }
       catch (LimitExceededException lex)
       {
           _logger.LogError(lex, $"Unable to add alarm {alarmName}. Alarm quota has
already been reached.");
       }

       return false;
   }
```

```
    /// <summary>
    /// Add specific email actions to a list of action strings for a CloudWatch
alarm.
    /// </summary>
    /// <param name="accountId">The AccountId for the alarm.</param>
    /// <param name="region">The region for the alarm.</param>
    /// <param name="emailTopicName">An Amazon Simple Notification Service (SNS)
topic for the alarm email.</param>
    /// <param name="alarmActions">Optional list of existing alarm actions to append
to.</param>
    /// <returns>A list of string actions for an alarm.</returns>
    public List<string> AddEmailAlarmAction(string accountId, string region,
        string emailTopicName, List<string>? alarmActions = null)
    {
        alarmActions ??= new List<string>();
        var snsAlarmAction = $"arn:aws:sns:{region}:{accountId}:{emailTopicName}";
        alarmActions.Add(snsAlarmAction);
        return alarmActions;
    }

    /// <summary>
    /// Describe the current alarms, optionally filtered by state.
    /// </summary>
    /// <param name="stateValue">Optional filter for alarm state.</param>
    /// <returns>The list of alarm data.</returns>
    public async Task<List<MetricAlarm>> DescribeAlarms(StateValue? stateValue =
null)
    {
        List<MetricAlarm> alarms = new List<MetricAlarm>();
        var paginatedDescribeAlarms = _amazonCloudWatch.Paginators.DescribeAlarms(
            new DescribeAlarmsRequest()
            {
                StateValue = stateValue
            });

        await foreach (var data in paginatedDescribeAlarms.MetricAlarms)
        {
            alarms.Add(data);
        }
        return alarms;
    }

    /// <summary>
    /// Describe the current alarms for a specific metric.
```

```
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricName">The name of the metric.</param>
    /// <returns>The list of alarm data.</returns>
    public async Task<List<MetricAlarm>> DescribeAlarmsForMetric(string
metricNamespace, string metricName)
    {
        var alarmsResult = await _amazonCloudWatch.DescribeAlarmsForMetricAsync(
            new DescribeAlarmsForMetricRequest()
            {
                Namespace = metricNamespace,
                MetricName = metricName
            });

        return alarmsResult.MetricAlarms ?? new List<MetricAlarm>();
    }

    /// <summary>
    /// Describe the history of an alarm for a number of days in the past.
    /// </summary>
    /// <param name="alarmName">The name of the alarm.</param>
    /// <param name="historyDays">The number of days in the past.</param>
    /// <returns>The list of alarm history data.</returns>
    public async Task<List<AlarmHistoryItem>> DescribeAlarmHistory(string alarmName,
int historyDays)
    {
        List<AlarmHistoryItem> alarmHistory = new List<AlarmHistoryItem>();
        var paginatedAlarmHistory =
_amazonCloudWatch.Paginators.DescribeAlarmHistory(
            new DescribeAlarmHistoryRequest()
            {
                AlarmName = alarmName,
                EndDateUtc = DateTime.UtcNow,
                HistoryItemType = HistoryItemType.StateUpdate,
                StartDateUtc = DateTime.UtcNow.AddDays(-historyDays)
            });

        await foreach (var data in paginatedAlarmHistory.AlarmHistoryItems)
        {
            alarmHistory.Add(data);
        }
        return alarmHistory;
    }
```

```
    /// <summary>
    /// Delete a list of alarms from CloudWatch.
    /// </summary>
    /// <param name="alarmNames">A list of names of alarms to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteAlarms(List<string> alarmNames)
    {
        var deleteAlarmsResult = await _amazonCloudWatch.DeleteAlarmsAsync(
            new DeleteAlarmsRequest()
            {
                AlarmNames = alarmNames
            });

        return deleteAlarmsResult.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Disable the actions for a list of alarms from CloudWatch.
    /// </summary>
    /// <param name="alarmNames">A list of names of alarms.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DisableAlarmActions(List<string> alarmNames)
    {
        var disableAlarmActionsResult = await
_amazonCloudWatch.DisableAlarmActionsAsync(
            new DisableAlarmActionsRequest()
            {
                AlarmNames = alarmNames
            });

        return disableAlarmActionsResult.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Enable the actions for a list of alarms from CloudWatch.
    /// </summary>
    /// <param name="alarmNames">A list of names of alarms.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> EnableAlarmActions(List<string> alarmNames)
    {
        var enableAlarmActionsResult = await
_amazonCloudWatch.EnableAlarmActionsAsync(
            new EnableAlarmActionsRequest()
            {
```

```
                AlarmNames = alarmNames
            });

        return enableAlarmActionsResult.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Add an anomaly detector for a single metric.
    /// </summary>
    /// <param name="anomalyDetector">A single metric anomaly detector.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> PutAnomalyDetector(SingleMetricAnomalyDetector
anomalyDetector)
    {
        var putAlarmDetectorResult = await
_amazonCloudWatch.PutAnomalyDetectorAsync(
            new PutAnomalyDetectorRequest()
            {
                SingleMetricAnomalyDetector = anomalyDetector
            });

        return putAlarmDetectorResult.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Describe anomaly detectors for a metric and namespace.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricName">The metric of the anomaly detectors.</param>
    /// <returns>The list of detectors.</returns>
    public async Task<List<AnomalyDetector>> DescribeAnomalyDetectors(string
metricNamespace, string metricName)
    {
        List<AnomalyDetector> detectors = new List<AnomalyDetector>();
        var paginatedDescribeAnomalyDetectors =
_amazonCloudWatch.Paginators.DescribeAnomalyDetectors(
            new DescribeAnomalyDetectorsRequest()
            {
                MetricName = metricName,
                Namespace = metricNamespace
            });

        await foreach (var data in
paginatedDescribeAnomalyDetectors.AnomalyDetectors)
```

```
        {
            detectors.Add(data);
        }

        return detectors;
    }

    /// <summary>
    /// Delete a single metric anomaly detector.
    /// </summary>
    /// <param name="anomalyDetector">The anomaly detector to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteAnomalyDetector(SingleMetricAnomalyDetector
 anomalyDetector)
    {
        var deleteAnomalyDetectorResponse = await
 _amazonCloudWatch.DeleteAnomalyDetectorAsync(
            new DeleteAnomalyDetectorRequest()
            {
                SingleMetricAnomalyDetector = anomalyDetector
            });

        return deleteAnomalyDetectorResponse.HttpStatusCode == HttpStatusCode.OK;
    }

    /// <summary>
    /// Delete a list of CloudWatch dashboards.
    /// </summary>
    /// <param name="dashboardNames">List of dashboard names to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteDashboards(List<string> dashboardNames)
    {
        var deleteDashboardsResponse = await
 _amazonCloudWatch.DeleteDashboardsAsync(
            new DeleteDashboardsRequest()
            {
                DashboardNames = dashboardNames
            });

        return deleteDashboardsResponse.HttpStatusCode == HttpStatusCode.OK;
    }
}
```

Exemplos de valores de settings.json para o cenário.

```
{
  "dashboardName": "example-new-dashboard",
  "exampleAlarmName": "example-metric-alarm",
  "accountId": "1234567890",
  "region": "us-east-1",
  "emailTopic": "Default_CloudWatch_Alarms_Topic",
  "customMetricNamespace": "example-namespace",
  "customMetricName": "example-custom-metric",
  "dashboardExampleBody": {
    "widgets": [
      {
        "height": 6,
        "width": 6,
        "y": 0,
        "x": 0,
        "type": "text",
        "properties": {
          "markdown": "# Code Example Dashboard \nThis dashboard was created by
 example code.\n"
        }
      },
      {
        "height": 8,
        "width": 8,
        "y": 0,
        "x": 6,
        "type": "metric",
        "properties": {
          "metrics": [
            [
              "AWS/Billing",
              "EstimatedCharges",
              "Currency",
              "USD",
              { "region": "us-east-1" }
            ]
          ],
          "view": "timeSeries",
          "region": "us-east-1",
          "stat": "Maximum",
          "period": 86400,
          "yAxis": {
```

```
            "left": {
              "min": 0,
              "max": 100
            }
          },
          "stacked": false,
          "title": "Estimated Billing",
          "setPeriodToTimeRange": false,
          "liveData": true,
          "sparkline": true,
          "trend": true
        }
      },
      {
        "height": 8,
        "width": 8,
        "y": 0,
        "x": 14,
        "type": "metric",
        "properties": {
          "metrics": [
            [ "AWS/Usage", "CallCount", "Type", "API", "Resource", "ListMetrics",
"Service", "CloudWatch", "Class", "None" ],
            [ "...", "GetMetricStatistics", ".", ".", ".", "." ],
            [ "...", "GetMetricData", ".", ".", ".", "." ],
            [ "...", "PutDashboard", ".", ".", ".", "." ],
            [ "...", "PutMetricData", ".", ".", ".", "." ]
          ],
          "view": "timeSeries",
          "yAxis": {
            "left": {
              "min": 0,
              "max": 200
            }
          },
          "stacked": false,
          "region": "us-east-1",
          "stat": "Sum",
          "period": 300,
          "title": "CloudWatch Usage",
          "setPeriodToTimeRange": false,
          "liveData": true,
          "sparkline": true,
          "trend": true
```

```
        }
      }
    ]
  }
}
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para .NET .
  - DeleteAlarms
  - DeleteAnomalyDetector
  - DeleteDashboards
  - DescribeAlarmHistory
  - DescribeAlarms
  - DescribeAlarmsForMetric
  - DescribeAnomalyDetectors
  - GetMetricData
  - GetMetricStatistics
  - GetMetricWidgetImage
  - ListMetrics
  - PutAnomalyDetector
  - PutDashboard
  - PutMetricAlarm
  - PutMetricData

Ações

`DeleteAlarms`

O código de exemplo a seguir mostra como usar `DeleteAlarms`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
/// <summary>
/// Delete a list of alarms from CloudWatch.
/// </summary>
/// <param name="alarmNames">A list of names of alarms to delete.</param>
/// <returns>True if successful.</returns>
public async Task<bool> DeleteAlarms(List<string> alarmNames)
{
    var deleteAlarmsResult = await _amazonCloudWatch.DeleteAlarmsAsync(
        new DeleteAlarmsRequest()
        {
            AlarmNames = alarmNames
        });

    return deleteAlarmsResult.HttpStatusCode == HttpStatusCode.OK;
}
```

- Para obter detalhes da API, consulte DeleteAlarmsa Referência AWS SDK para .NET da API.

`DeleteAnomalyDetector`

O código de exemplo a seguir mostra como usar `DeleteAnomalyDetector`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
/// <summary>
```

```
    /// Delete a single metric anomaly detector.
    /// </summary>
    /// <param name="anomalyDetector">The anomaly detector to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteAnomalyDetector(SingleMetricAnomalyDetector
anomalyDetector)
    {
        var deleteAnomalyDetectorResponse = await
_amazonCloudWatch.DeleteAnomalyDetectorAsync(
            new DeleteAnomalyDetectorRequest()
            {
                SingleMetricAnomalyDetector = anomalyDetector
            });

        return deleteAnomalyDetectorResponse.HttpStatusCode == HttpStatusCode.OK;
    }
```

- Para obter detalhes da API, consulte DeleteAnomalyDetectora Referência AWS SDK para .NET da API.

`DeleteDashboards`

O código de exemplo a seguir mostra como usar `DeleteDashboards`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Delete a list of CloudWatch dashboards.
    /// </summary>
    /// <param name="dashboardNames">List of dashboard names to delete.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteDashboards(List<string> dashboardNames)
    {
```

```
        var deleteDashboardsResponse = await
_amazonCloudWatch.DeleteDashboardsAsync(
            new DeleteDashboardsRequest()
            {
                DashboardNames = dashboardNames
            });

        return deleteDashboardsResponse.HttpStatusCode == HttpStatusCode.OK;
    }
```

- Para obter detalhes da API, consulte DeleteDashboardsa Referência AWS SDK para .NET da API.

`DescribeAlarmHistory`

O código de exemplo a seguir mostra como usar `DescribeAlarmHistory`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Describe the history of an alarm for a number of days in the past.
    /// </summary>
    /// <param name="alarmName">The name of the alarm.</param>
    /// <param name="historyDays">The number of days in the past.</param>
    /// <returns>The list of alarm history data.</returns>
    public async Task<List<AlarmHistoryItem>> DescribeAlarmHistory(string alarmName,
int historyDays)
    {
        List<AlarmHistoryItem> alarmHistory = new List<AlarmHistoryItem>();
        var paginatedAlarmHistory =
_amazonCloudWatch.Paginators.DescribeAlarmHistory(
            new DescribeAlarmHistoryRequest()
            {
                AlarmName = alarmName,
```

```
                EndDateUtc = DateTime.UtcNow,
                HistoryItemType = HistoryItemType.StateUpdate,
                StartDateUtc = DateTime.UtcNow.AddDays(-historyDays)
            });

        await foreach (var data in paginatedAlarmHistory.AlarmHistoryItems)
        {
            alarmHistory.Add(data);
        }
        return alarmHistory;
    }
```

- Para obter detalhes da API, consulte DescribeAlarmHistorya Referência AWS SDK para .NET da API.

`DescribeAlarms`

O código de exemplo a seguir mostra como usar `DescribeAlarms`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Describe the current alarms, optionally filtered by state.
    /// </summary>
    /// <param name="stateValue">Optional filter for alarm state.</param>
    /// <returns>The list of alarm data.</returns>
    public async Task<List<MetricAlarm>> DescribeAlarms(StateValue? stateValue =
null)
    {
        List<MetricAlarm> alarms = new List<MetricAlarm>();
        var paginatedDescribeAlarms = _amazonCloudWatch.Paginators.DescribeAlarms(
            new DescribeAlarmsRequest()
            {
                StateValue = stateValue
```

```
            });

        await foreach (var data in paginatedDescribeAlarms.MetricAlarms)
        {
            alarms.Add(data);
        }
        return alarms;
    }
```

- Para obter detalhes da API, consulte DescribeAlarmsa Referência AWS SDK para .NET da API.

`DescribeAlarmsForMetric`

O código de exemplo a seguir mostra como usar `DescribeAlarmsForMetric`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Describe the current alarms for a specific metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricName">The name of the metric.</param>
    /// <returns>The list of alarm data.</returns>
    public async Task<List<MetricAlarm>> DescribeAlarmsForMetric(string
metricNamespace, string metricName)
    {
        var alarmsResult = await _amazonCloudWatch.DescribeAlarmsForMetricAsync(
            new DescribeAlarmsForMetricRequest()
            {
                Namespace = metricNamespace,
                MetricName = metricName
            });
```

```
        return alarmsResult.MetricAlarms ?? new List<MetricAlarm>();
    }
```

- Para obter detalhes da API, consulte DescribeAlarmsForMetrica Referência AWS SDK para .NET da API.

`DescribeAnomalyDetectors`

O código de exemplo a seguir mostra como usar `DescribeAnomalyDetectors`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Describe anomaly detectors for a metric and namespace.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricName">The metric of the anomaly detectors.</param>
    /// <returns>The list of detectors.</returns>
    public async Task<List<AnomalyDetector>> DescribeAnomalyDetectors(string
metricNamespace, string metricName)
    {
        List<AnomalyDetector> detectors = new List<AnomalyDetector>();
        var paginatedDescribeAnomalyDetectors =
_amazonCloudWatch.Paginators.DescribeAnomalyDetectors(
            new DescribeAnomalyDetectorsRequest()
            {
                MetricName = metricName,
                Namespace = metricNamespace
            });

        await foreach (var data in
paginatedDescribeAnomalyDetectors.AnomalyDetectors)
        {
            detectors.Add(data);
```

```
        }

        return detectors;
    }
```

- Para obter detalhes da API, consulte DescribeAnomalyDetectorsa Referência AWS SDK para .NET da API.

`DisableAlarmActions`

O código de exemplo a seguir mostra como usar `DisableAlarmActions`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Disable the actions for a list of alarms from CloudWatch.
    /// </summary>
    /// <param name="alarmNames">A list of names of alarms.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DisableAlarmActions(List<string> alarmNames)
    {
        var disableAlarmActionsResult = await
_amazonCloudWatch.DisableAlarmActionsAsync(
            new DisableAlarmActionsRequest()
            {
                AlarmNames = alarmNames
            });

        return disableAlarmActionsResult.HttpStatusCode == HttpStatusCode.OK;
    }
```

- Para obter detalhes da API, consulte DisableAlarmActionsa Referência AWS SDK para .NET da API.

`EnableAlarmActions`

O código de exemplo a seguir mostra como usar `EnableAlarmActions`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Enable the actions for a list of alarms from CloudWatch.
    /// </summary>
    /// <param name="alarmNames">A list of names of alarms.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> EnableAlarmActions(List<string> alarmNames)
    {
        var enableAlarmActionsResult = await
_amazonCloudWatch.EnableAlarmActionsAsync(
            new EnableAlarmActionsRequest()
            {
                AlarmNames = alarmNames
            });

        return enableAlarmActionsResult.HttpStatusCode == HttpStatusCode.OK;
    }
```

- Para obter detalhes da API, consulte EnableAlarmActionsa Referência AWS SDK para .NET da API.

`GetDashboard`

O código de exemplo a seguir mostra como usar `GetDashboard`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
/// <summary>
/// Get information on a dashboard.
/// </summary>
/// <param name="dashboardName">The name of the dashboard.</param>
/// <returns>A JSON object with dashboard information.</returns>
public async Task<string> GetDashboard(string dashboardName)
{
    var dashboardResponse = await _amazonCloudWatch.GetDashboardAsync(
        new GetDashboardRequest()
        {
            DashboardName = dashboardName
        });

    return dashboardResponse.DashboardBody;
}
```

- Para obter detalhes da API, consulte GetDashboarda Referência AWS SDK para .NET da API.

`GetMetricData`

O código de exemplo a seguir mostra como usar `GetMetricData`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Get data for CloudWatch metrics.
    /// </summary>
    /// <param name="minutesOfData">The number of minutes of data to include.</
param>
    /// <param name="useDescendingTime">True to return the data descending by
 time.</param>
    /// <param name="endDateUtc">The end date for the data, in UTC.</param>
    /// <param name="maxDataPoints">The maximum data points to include.</param>
    /// <param name="dataQueries">Optional data queries to include.</param>
    /// <returns>A list of the requested metric data.</returns>
    public async Task<List<MetricDataResult>> GetMetricData(int minutesOfData, bool
 useDescendingTime, DateTime? endDateUtc = null,
        int maxDataPoints = 0, List<MetricDataQuery>? dataQueries = null)
    {
        var metricData = new List<MetricDataResult>();
        // If no end time is provided, use the current time for the end time.
        endDateUtc ??= DateTime.UtcNow;
        var timeZoneOffset =
 TimeZoneInfo.Local.GetUtcOffset(endDateUtc.Value.ToLocalTime());
        var startTimeUtc = endDateUtc.Value.AddMinutes(-minutesOfData);
        // The timezone string should be in the format +0000, so use the timezone
 offset to format it correctly.
        var timeZoneString = $"{timeZoneOffset.Hours:D2}
{timeZoneOffset.Minutes:D2}";
        // Add the plus sign for positive offsets.
        timeZoneString = timeZoneString.StartsWith('-') ? timeZoneString : "+" +
 timeZoneString;
        var paginatedMetricData = _amazonCloudWatch.Paginators.GetMetricData(
            new GetMetricDataRequest()
            {
                StartTimeUtc = startTimeUtc,
                EndTimeUtc = endDateUtc.Value,
                LabelOptions = new LabelOptions { Timezone = timeZoneString },
                ScanBy = useDescendingTime ? ScanBy.TimestampDescending :
 ScanBy.TimestampAscending,
                MaxDatapoints = maxDataPoints,
                MetricDataQueries = dataQueries,
            });

        if (paginatedMetricData.MetricDataResults != null)
        {
            await foreach (var data in paginatedMetricData.MetricDataResults)
```

```
            {
                metricData.Add(data);
            }
        }

        return metricData;
    }
```

- Para obter detalhes da API, consulte GetMetricDataa Referência AWS SDK para .NET da API.

`GetMetricStatistics`

O código de exemplo a seguir mostra como usar `GetMetricStatistics`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Get billing statistics using a call to a wrapper class.
    /// </summary>
    /// <returns>A collection of billing statistics.</returns>
    private static async Task<List<Datapoint>> SetupBillingStatistics()
    {
        // Make a request for EstimatedCharges with a period of one day for the past
seven days.
        var billingStatistics = await _cloudWatchWrapper.GetMetricStatistics(
            "AWS/Billing",
            "EstimatedCharges",
            new List<string>() { "Maximum" },
            new List<Dimension>() { new Dimension { Name = "Currency", Value =
"USD" } },
            7,
            86400);

        billingStatistics = billingStatistics.OrderBy(n => n.Timestamp).ToList();
```

```
        return billingStatistics;
    }

    /// <summary>
    /// Wrapper to get statistics for a specific CloudWatch metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricName">The name of the metric.</param>
    /// <param name="statistics">The list of statistics to include.</param>
    /// <param name="dimensions">The list of dimensions to include.</param>
    /// <param name="days">The number of days in the past to include.</param>
    /// <param name="period">The period for the data.</param>
    /// <returns>A list of DataPoint objects for the statistics.</returns>
    public async Task<List<Datapoint>> GetMetricStatistics(string metricNamespace,
        string metricName, List<string> statistics, List<Dimension> dimensions, int
days, int period)
    {
        var metricStatistics = await _amazonCloudWatch.GetMetricStatisticsAsync(
            new GetMetricStatisticsRequest()
            {
                Namespace = metricNamespace,
                MetricName = metricName,
                Dimensions = dimensions,
                Statistics = statistics,
                StartTimeUtc = DateTime.UtcNow.AddDays(-days),
                EndTimeUtc = DateTime.UtcNow,
                Period = period
            });

        return metricStatistics.Datapoints ?? new List<Datapoint>();
    }
```

- Para obter detalhes da API, consulte GetMetricStatisticsa Referência AWS SDK para .NET da API.

`GetMetricWidgetImage`

O código de exemplo a seguir mostra como usar `GetMetricWidgetImage`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Get an image for a metric graphed over time.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metric">The name of the metric.</param>
    /// <param name="stat">The name of the stat to chart.</param>
    /// <param name="period">The period to use for the chart.</param>
    /// <returns>A memory stream for the chart image.</returns>
    public async Task<MemoryStream> GetTimeSeriesMetricImage(string metricNamespace,
string metric, string stat, int period)
    {
        var metricImageWidget = new
        {
            title = "Example Metric Graph",
            view = "timeSeries",
            stacked = false,
            period = period,
            width = 1400,
            height = 600,
            metrics = new List<List<object>>
                { new() { metricNamespace, metric, new { stat } } }
        };

        var metricImageWidgetString = JsonSerializer.Serialize(metricImageWidget);
        var imageResponse = await _amazonCloudWatch.GetMetricWidgetImageAsync(
            new GetMetricWidgetImageRequest()
            {
                MetricWidget = metricImageWidgetString
            });

        return imageResponse.MetricWidgetImage;
    }

    /// <summary>
```

```
    /// Save a metric image to a file.
    /// </summary>
    /// <param name="memoryStream">The MemoryStream for the metric image.</param>
    /// <param name="metricName">The name of the metric.</param>
    /// <returns>The path to the file.</returns>
    public string SaveMetricImage(MemoryStream memoryStream, string metricName)
    {
        var metricFileName = $"{metricName}_{DateTime.Now.Ticks}.png";
        using var sr = new StreamReader(memoryStream);
        // Writes the memory stream to a file.
        File.WriteAllBytes(metricFileName, memoryStream.ToArray());
        var filePath = Path.Join(AppDomain.CurrentDomain.BaseDirectory,
            metricFileName);
        return filePath;
    }
```

- Para obter detalhes da API, consulte GetMetricWidgetImagea Referência AWS SDK para .NET da API.

`ListDashboards`

O código de exemplo a seguir mostra como usar `ListDashboards`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Get a list of dashboards.
    /// </summary>
    /// <returns>A list of DashboardEntry objects.</returns>
    public async Task<List<DashboardEntry>> ListDashboards()
    {
        var results = new List<DashboardEntry>();
        var paginateDashboards = _amazonCloudWatch.Paginators.ListDashboards(
            new ListDashboardsRequest());
```

```
        // Get the entire list using the paginator.
        await foreach (var data in paginateDashboards.DashboardEntries)
        {
            results.Add(data);
        }

        return results;
    }
```

- Para obter detalhes da API, consulte ListDashboardsa Referência AWS SDK para .NET da API.

`ListMetrics`

O código de exemplo a seguir mostra como usar `ListMetrics`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// List metrics available, optionally within a namespace.
    /// </summary>
    /// <param name="metricNamespace">Optional CloudWatch namespace to use when
listing metrics.</param>
    /// <param name="filter">Optional dimension filter.</param>
    /// <param name="metricName">Optional metric name filter.</param>
    /// <returns>The list of metrics.</returns>
    public async Task<List<Metric>> ListMetrics(string? metricNamespace = null,
DimensionFilter? filter = null, string? metricName = null)
    {
        var results = new List<Metric>();
        var paginateMetrics = _amazonCloudWatch.Paginators.ListMetrics(
            new ListMetricsRequest
            {
                Namespace = metricNamespace,
                Dimensions = filter != null ? new List<DimensionFilter> { filter } :
null,
```

```
                MetricName = metricName
            });
        // Get the entire list using the paginator.
        await foreach (var metric in paginateMetrics.Metrics)
        {
            results.Add(metric);
        }

        return results;
    }
```

- Para obter detalhes da API, consulte ListMetricsa Referência AWS SDK para .NET da API.

`PutAnomalyDetector`

O código de exemplo a seguir mostra como usar `PutAnomalyDetector`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Add an anomaly detector for a single metric.
    /// </summary>
    /// <param name="anomalyDetector">A single metric anomaly detector.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> PutAnomalyDetector(SingleMetricAnomalyDetector
anomalyDetector)
    {
        var putAlarmDetectorResult = await
_amazonCloudWatch.PutAnomalyDetectorAsync(
            new PutAnomalyDetectorRequest()
            {
                SingleMetricAnomalyDetector = anomalyDetector
            });

        return putAlarmDetectorResult.HttpStatusCode == HttpStatusCode.OK;
```

```
    }
```

- Para obter detalhes da API, consulte PutAnomalyDetectora Referência AWS SDK para .NET da API.

`PutDashboard`

O código de exemplo a seguir mostra como usar `PutDashboard`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
/// <summary>
/// Set up a dashboard using a call to the wrapper class.
/// </summary>
/// <param name="customMetricNamespace">The metric namespace.</param>
/// <param name="customMetricName">The metric name.</param>
/// <param name="dashboardName">The name of the dashboard.</param>
/// <returns>A list of validation messages.</returns>
private static async Task<List<DashboardValidationMessage>> SetupDashboard(
    string customMetricNamespace, string customMetricName, string dashboardName)
{
    // Get the dashboard model from configuration.
    var newDashboard = new DashboardModel();
    _configuration.GetSection("dashboardExampleBody").Bind(newDashboard);

    // Add a new metric to the dashboard.
    newDashboard.Widgets.Add(new Widget
    {
        Height = 8,
        Width = 8,
        Y = 8,
        X = 0,
        Type = "metric",
```

```
            Properties = new Properties
            {
                Metrics = new List<List<object>>
                    { new() { customMetricNamespace, customMetricName } },
                View = "timeSeries",
                Region = "us-east-1",
                Stat = "Sum",
                Period = 86400,
                YAxis = new YAxis { Left = new Left { Min = 0, Max = 100 } },
                Title = "Custom Metric Widget",
                LiveData = true,
                Sparkline = true,
                Trend = true,
                Stacked = false,
                SetPeriodToTimeRange = false
            }
        });

        var newDashboardString = JsonSerializer.Serialize(newDashboard,
            new JsonSerializerOptions
            { DefaultIgnoreCondition = JsonIgnoreCondition.WhenWritingNull });
        var validationMessages =
            await _cloudWatchWrapper.PutDashboard(dashboardName,
 newDashboardString);

        return validationMessages;
    }

    /// <summary>
    /// Wrapper to create or add to a dashboard with metrics.
    /// </summary>
    /// <param name="dashboardName">The name for the dashboard.</param>
    /// <param name="dashboardBody">The metric data in JSON for the dashboard.</
param>
    /// <returns>A list of validation messages for the dashboard.</returns>
    public async Task<List<DashboardValidationMessage>> PutDashboard(string
 dashboardName,
        string dashboardBody)
    {
        // Updating a dashboard replaces all contents.
        // Best practice is to include a text widget indicating this dashboard was
 created programmatically.
        var dashboardResponse = await _amazonCloudWatch.PutDashboardAsync(
            new PutDashboardRequest()
```

```
            {
                DashboardName = dashboardName,
                DashboardBody = dashboardBody
            });

        return dashboardResponse.DashboardValidationMessages ?? new
List<DashboardValidationMessage>();
    }
```

- Para obter detalhes da API, consulte PutDashboarda Referência AWS SDK para .NET da API.

`PutMetricAlarm`

O código de exemplo a seguir mostra como usar `PutMetricAlarm`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Add a metric alarm to send an email when the metric passes a threshold.
    /// </summary>
    /// <param name="alarmDescription">A description of the alarm.</param>
    /// <param name="alarmName">The name for the alarm.</param>
    /// <param name="comparison">The type of comparison to use.</param>
    /// <param name="metricName">The name of the metric for the alarm.</param>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="threshold">The threshold value for the alarm.</param>
    /// <param name="alarmActions">Optional actions to execute when in an alarm
state.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> PutMetricEmailAlarm(string alarmDescription, string
alarmName, ComparisonOperator comparison,
        string metricName, string metricNamespace, double threshold, List<string>
alarmActions = null!)
    {
```

```
        try
        {
            var putEmailAlarmResponse = await _amazonCloudWatch.PutMetricAlarmAsync(
                new PutMetricAlarmRequest()
                {
                    AlarmActions = alarmActions,
                    AlarmDescription = alarmDescription,
                    AlarmName = alarmName,
                    ComparisonOperator = comparison,
                    Threshold = threshold,
                    Namespace = metricNamespace,
                    MetricName = metricName,
                    EvaluationPeriods = 1,
                    Period = 10,
                    Statistic = new Statistic("Maximum"),
                    DatapointsToAlarm = 1,
                    TreatMissingData = "ignore"
                });
            return putEmailAlarmResponse.HttpStatusCode == HttpStatusCode.OK;
        }
        catch (LimitExceededException lex)
        {
            _logger.LogError(lex, $"Unable to add alarm {alarmName}. Alarm quota has
already been reached.");
        }

        return false;
    }

    /// <summary>
    /// Add specific email actions to a list of action strings for a CloudWatch
alarm.
    /// </summary>
    /// <param name="accountId">The AccountId for the alarm.</param>
    /// <param name="region">The region for the alarm.</param>
    /// <param name="emailTopicName">An Amazon Simple Notification Service (SNS)
topic for the alarm email.</param>
    /// <param name="alarmActions">Optional list of existing alarm actions to append
to.</param>
    /// <returns>A list of string actions for an alarm.</returns>
    public List<string> AddEmailAlarmAction(string accountId, string region,
        string emailTopicName, List<string>? alarmActions = null)
    {
        alarmActions ??= new List<string>();
```

```
        var snsAlarmAction = $"arn:aws:sns:{region}:{accountId}:{emailTopicName}";
        alarmActions.Add(snsAlarmAction);
        return alarmActions;
    }
```

- Para obter detalhes da API, consulte PutMetricAlarma Referência AWS SDK para .NET da API.

`PutMetricData`

O código de exemplo a seguir mostra como usar `PutMetricData`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Add some metric data using a call to a wrapper class.
    /// </summary>
    /// <param name="customMetricName">The metric name.</param>
    /// <param name="customMetricNamespace">The metric namespace.</param>
    /// <returns></returns>
    private static async Task<List<MetricDatum>> PutRandomMetricData(string
customMetricName,
        string customMetricNamespace)
    {
        List<MetricDatum> customData = new List<MetricDatum>();
        Random rnd = new Random();

        // Add 10 random values up to 100, starting with a timestamp 15 minutes in
the past.
        var utcNowMinus15 = DateTime.UtcNow.AddMinutes(-15);
        for (int i = 0; i < 10; i++)
        {
            var metricValue = rnd.Next(0, 100);
            customData.Add(
                new MetricDatum
```

```
                {
                    MetricName = customMetricName,
                    Value = metricValue,
                    TimestampUtc = utcNowMinus15.AddMinutes(i)
                }
            );
        }

        await _cloudWatchWrapper.PutMetricData(customMetricNamespace, customData);
        return customData;
    }

    /// <summary>
    /// Wrapper to add metric data to a CloudWatch metric.
    /// </summary>
    /// <param name="metricNamespace">The namespace of the metric.</param>
    /// <param name="metricData">A data object for the metric data.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> PutMetricData(string metricNamespace,
        List<MetricDatum> metricData)
    {
        var putDataResponse = await _amazonCloudWatch.PutMetricDataAsync(
            new PutMetricDataRequest()
            {
                MetricData = metricData,
                Namespace = metricNamespace,
            });

        return putDataResponse.HttpStatusCode == HttpStatusCode.OK;
    }
```

- Para obter detalhes da API, consulte PutMetricDataa Referência AWS SDK para .NET da API.

CloudWatch Exemplos de registros usando SDK para .NET (v4)

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET (v4) com o CloudWatch Logs.

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar perfis de serviço individuais, você pode ver as ações no contexto em seus cenários relacionados.

Cenários são exemplos de código que mostram como realizar tarefas específicas chamando várias funções dentro de um serviço ou combinadas com outros Serviços da AWS.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- Ações
- Cenários

Ações

`GetQueryResults`

O código de exemplo a seguir mostra como usar `GetQueryResults`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Gets the results of a CloudWatch Logs Insights query.
    /// </summary>
    /// <param name="queryId">The ID of the query.</param>
    /// <returns>The query results response.</returns>
    public async Task<GetQueryResultsResponse?> GetQueryResultsAsync(string queryId)
    {
        try
        {
            var request = new GetQueryResultsRequest
            {
                QueryId = queryId
            };

            var response = await
_amazonCloudWatchLogs.GetQueryResultsAsync(request);
```

```
            return response;
        }
        catch (ResourceNotFoundException ex)
        {
            _logger.LogError($"Query not found: {ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"An error occurred while getting query results:
 {ex.Message}");
            return null;
        }
    }
```

- Para obter detalhes da API, consulte GetQueryResultsa Referência AWS SDK para .NET da API.

`StartQuery`

O código de exemplo a seguir mostra como usar `StartQuery`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Starts a CloudWatch Logs Insights query.
    /// </summary>
    /// <param name="logGroupName">The name of the log group to query.</param>
    /// <param name="queryString">The CloudWatch Logs Insights query string.</param>
    /// <param name="startTime">The start time for the query (seconds since
 epoch).</param>
    /// <param name="endTime">The end time for the query (seconds since epoch).</
param>
    /// <param name="limit">The maximum number of results to return.</param>
```

```
    /// <returns>The query ID if successful, null otherwise.</returns>
    public async Task<string?> StartQueryAsync(
        string logGroupName,
        string queryString,
        long startTime,
        long endTime,
        int limit = 10000)
    {
        try
        {
            var request = new StartQueryRequest
            {
                LogGroupName = logGroupName,
                QueryString = queryString,
                StartTime = startTime,
                EndTime = endTime,
                Limit = limit
            };

            var response = await _amazonCloudWatchLogs.StartQueryAsync(request);
            return response.QueryId;
        }
        catch (InvalidParameterException ex)
        {
            _logger.LogError($"Invalid parameter for query: {ex.Message}");
            return null;
        }
        catch (ResourceNotFoundException ex)
        {
            _logger.LogError($"Log group not found: {ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"An error occurred while starting query:
{ex.Message}");
            return null;
        }
    }
```

- Para obter detalhes da API, consulte StartQuerya Referência AWS SDK para .NET da API.

Cenários

Executar uma consulta grande

O exemplo de código a seguir mostra como usar o CloudWatch Logs para consultar mais de 10.000 registros.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Esse é o fluxo de trabalho principal que demonstra o grande cenário de consultas.

```
using System.Diagnostics;
using System.Text.RegularExpressions;
using Amazon.CloudFormation;
using Amazon.CloudFormation.Model;
using Amazon.CloudWatchLogs;
using Amazon.CloudWatchLogs.Model;
using CloudWatchLogsActions;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;

namespace CloudWatchLogsScenario;

public class LargeQueryWorkflow
{
    /*
    Before running this .NET code example, set up your development environment,
 including your credentials.
    This .NET code example performs the following tasks for the CloudWatch Logs
 Large Query workflow:

    1. Prepare the Application:
       - Prompt the user to deploy CloudFormation stack and generate sample logs.
       - Deploy the CloudFormation template for resource creation.
       - Generate 50,000 sample log entries using CloudWatch Logs API.
```

```
       - Wait 5 minutes for logs to be fully ingested.

    2. Execute Large Query:
       - Perform recursive queries to retrieve all logs using binary search.
       - Display progress for each query executed.
       - Show total execution time and logs found.

    3. Clean up:
       - Prompt the user to delete the CloudFormation stack and all resources.
       - Destroy the CloudFormation stack and wait until removed.
    */

    public static ILogger<LargeQueryWorkflow> _logger = null!;
    public static CloudWatchLogsWrapper _wrapper = null!;
    public static IAmazonCloudFormation _amazonCloudFormation = null!;

    private static string _logGroupName = "/workflows/cloudwatch-logs/large-query";
    private static string _logStreamName = "stream1";
    private static long _queryStartDate;
    private static long _queryEndDate;

    public static bool _interactive = true;
    public static string _stackName = "CloudWatchLargeQueryStack";
    private static string _stackResourcePath = "../../../../../../../scenarios/
features/cloudwatch_logs_large_query/resources/stack.yaml";

    public static async Task Main(string[] args)
    {
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter("Microsoft", LogLevel.Information))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonCloudWatchLogs>()
                    .AddAWSService<IAmazonCloudFormation>()
                    .AddTransient<CloudWatchLogsWrapper>()
            )
            .Build();

        if (_interactive)
        {
            _logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
                .CreateLogger<LargeQueryWorkflow>();
```

```
            _wrapper = host.Services.GetRequiredService<CloudWatchLogsWrapper>();
            _amazonCloudFormation =
host.Services.GetRequiredService<IAmazonCloudFormation>();
        }

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Welcome to the CloudWatch Logs Large Query Scenario.");
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("This scenario demonstrates how to perform large-scale
queries on");
        Console.WriteLine("CloudWatch Logs using recursive binary search to retrieve
more than");
        Console.WriteLine("the 10,000 result limit.");
        Console.WriteLine();

        try
        {
            Console.WriteLine(new string('-', 80));
            var prepareSuccess = await PrepareApplication();
            Console.WriteLine(new string('-', 80));

            if (prepareSuccess)
            {
                Console.WriteLine(new string('-', 80));
                await ExecuteLargeQuery();
                Console.WriteLine(new string('-', 80));
            }

            Console.WriteLine(new string('-', 80));
            await Cleanup();
            Console.WriteLine(new string('-', 80));
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "There was a problem with the scenario, initiating
cleanup...");
            _interactive = false;
            await Cleanup();
        }

        Console.WriteLine("CloudWatch Logs Large Query scenario completed.");
    }

    /// <summary>
```

```
    /// Runs the scenario workflow. Used for testing.
    /// </summary>
    public static async Task RunScenario()
    {
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Welcome to the CloudWatch Logs Large Query Scenario.");
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("This scenario demonstrates how to perform large-scale
queries on");
        Console.WriteLine("CloudWatch Logs using recursive binary search to retrieve
more than");
        Console.WriteLine("the 10,000 result limit.");
        Console.WriteLine();

        try
        {
            Console.WriteLine(new string('-', 80));
            var prepareSuccess = await PrepareApplication();
            Console.WriteLine(new string('-', 80));

            if (prepareSuccess)
            {
                Console.WriteLine(new string('-', 80));
                await ExecuteLargeQuery();
                Console.WriteLine(new string('-', 80));
            }

            Console.WriteLine(new string('-', 80));
            await Cleanup();
            Console.WriteLine(new string('-', 80));
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "There was a problem with the scenario, initiating
cleanup...");
            _interactive = false;
            await Cleanup();
        }

        Console.WriteLine("CloudWatch Logs Large Query scenario completed.");
    }

    /// <summary>
    /// Prepares the application by creating the necessary resources.
```

```
    /// </summary>
    /// <returns>True if the application was prepared successfully.</returns>
    public static async Task<bool> PrepareApplication()
    {
        Console.WriteLine("Preparing the application...");
        Console.WriteLine();

        try
        {
            var deployStack = !_interactive || GetYesNoResponse(
                "Would you like to deploy the CloudFormation stack and generate
sample logs? (y/n) ");

            if (deployStack)
            {
                if (_interactive)
                {
                    Console.Write(
                        $"Enter a path for the CloudFormation stack resource .yaml
file (or press Enter for default '{_stackResourcePath}'): ");
                    string? inputPath = Console.ReadLine();
                    if (!string.IsNullOrWhiteSpace(inputPath))
                    {
                        _stackResourcePath = inputPath;
                    }
                }

                _stackName = PromptUserForStackName();

                var deploySuccess = await DeployCloudFormationStack(_stackName);

                if (deploySuccess)
                {
                    Console.WriteLine();
                    Console.WriteLine("Generating 50,000 sample log entries...");
                    var generateSuccess = await GenerateSampleLogs();

                    if (generateSuccess)
                    {
                        Console.WriteLine();
                        Console.WriteLine("Sample logs created. Waiting 5 minutes
for logs to be fully ingested...");
                        await WaitWithCountdown(300);
```

```
                    Console.WriteLine("Application preparation complete.");
                    return true;
                }
            }
        }
        else
        {
            _logGroupName = PromptUserForInput("Enter the log group name ",
_logGroupName);
            _logStreamName = PromptUserForInput("Enter the log stream name ",
_logStreamName);

            var startDateMs = PromptUserForLong("Enter the query start date
(milliseconds since epoch): ");
            var endDateMs = PromptUserForLong("Enter the query end date
(milliseconds since epoch): ");

            _queryStartDate = startDateMs / 1000;
            _queryEndDate = endDateMs / 1000;

            Console.WriteLine("Application preparation complete.");
            return true;
        }
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "An error occurred while preparing the
application.");
    }

    Console.WriteLine("Application preparation failed.");
    return false;
}

/// <summary>
/// Deploys the CloudFormation stack with the necessary resources.
/// </summary>
/// <param name="stackName">The name of the CloudFormation stack.</param>
/// <returns>True if the stack was deployed successfully.</returns>
private static async Task<bool> DeployCloudFormationStack(string stackName)
{
    Console.WriteLine($"\nDeploying CloudFormation stack: {stackName}");

    try
```

```
        {
            var request = new CreateStackRequest
            {
                StackName = stackName,
                TemplateBody = await File.ReadAllTextAsync(_stackResourcePath)
            };

            var response = await _amazonCloudFormation.CreateStackAsync(request);

            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine($"CloudFormation stack creation started:
{stackName}");

                bool stackCreated = await WaitForStackCompletion(response.StackId);

                if (stackCreated)
                {
                    Console.WriteLine("CloudFormation stack created successfully.");
                    return true;
                }
                else
                {
                    _logger.LogError($"CloudFormation stack creation failed:
{stackName}");
                    return false;
                }
            }
            else
            {
                _logger.LogError($"Failed to create CloudFormation stack:
{stackName}");
                return false;
            }
        }
        catch (AlreadyExistsException)
        {
            _logger.LogWarning($"CloudFormation stack '{stackName}' already exists.
Please provide a unique name.");
            var newStackName = PromptUserForStackName();
            return await DeployCloudFormationStack(newStackName);
        }
        catch (Exception ex)
        {
```

```
            _logger.LogError(ex, $"An error occurred while deploying the
CloudFormation stack: {stackName}");
            return false;
        }
    }

    /// <summary>
    /// Waits for the CloudFormation stack to be in the CREATE_COMPLETE state.
    /// </summary>
    /// <param name="stackId">The ID of the CloudFormation stack.</param>
    /// <returns>True if the stack was created successfully.</returns>
    private static async Task<bool> WaitForStackCompletion(string stackId)
    {
        int retryCount = 0;
        const int maxRetries = 30;
        const int retryDelay = 10000;

        while (retryCount < maxRetries)
        {
            var describeStacksRequest = new DescribeStacksRequest
            {
                StackName = stackId
            };

            var describeStacksResponse = await
_amazonCloudFormation.DescribeStacksAsync(describeStacksRequest);

            if (describeStacksResponse.Stacks.Count > 0)
            {
                if (describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.CREATE_COMPLETE)
                {
                    return true;
                }
                if (describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.CREATE_FAILED ||
                    describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.ROLLBACK_COMPLETE)
                {
                    return false;
                }
            }
```

```
            Console.WriteLine("Waiting for CloudFormation stack creation to
complete...");
            await Task.Delay(retryDelay);
            retryCount++;
        }

        _logger.LogError("Timed out waiting for CloudFormation stack creation to
complete.");
        return false;
    }

    /// <summary>
    /// Generates sample logs directly using CloudWatch Logs API.
    /// Creates 50,000 log entries spanning 5 minutes.
    /// </summary>
    /// <returns>True if logs were generated successfully.</returns>
    private static async Task<bool> GenerateSampleLogs()
    {
        const int totalEntries = 50000;
        const int entriesPerBatch = 10000;
        const int fiveMinutesMs = 5 * 60 * 1000;

        try
        {
            // Calculate timestamps
            var startTimeMs = DateTimeOffset.UtcNow.ToUnixTimeMilliseconds();
            var timestampIncrement = fiveMinutesMs / totalEntries;

            Console.WriteLine($"Generating {totalEntries} log entries...");

            var entryCount = 0;
            var currentTimestamp = startTimeMs;
            var numBatches = totalEntries / entriesPerBatch;

            // Generate and upload logs in batches
            for (int batchNum = 0; batchNum < numBatches; batchNum++)
            {
                var logEvents = new List<InputLogEvent>();

                for (int i = 0; i < entriesPerBatch; i++)
                {
                    logEvents.Add(new InputLogEvent
                    {
```

```
                        Timestamp =
 DateTimeOffset.FromUnixTimeMilliseconds(currentTimestamp).UtcDateTime,
                        Message = $"Entry {entryCount}"
                    });

                    entryCount++;
                    currentTimestamp += timestampIncrement;
                }

                // Upload batch
                var success = await _wrapper.PutLogEventsAsync(_logGroupName,
 _logStreamName, logEvents);
                if (!success)
                {
                    _logger.LogError($"Failed to upload batch {batchNum + 1}/
{numBatches}");
                    return false;
                }

                Console.WriteLine($"Uploaded batch {batchNum + 1}/{numBatches}");
            }

            // Set query date range (convert milliseconds to seconds for query API)
            _queryStartDate = startTimeMs / 1000;
            _queryEndDate = (currentTimestamp - timestampIncrement) / 1000;

            Console.WriteLine($"Query start date:
 {DateTimeOffset.FromUnixTimeSeconds(_queryStartDate):yyyy-MM-ddTHH:mm:ss.fffZ}");
            Console.WriteLine($"Query end date:
 {DateTimeOffset.FromUnixTimeSeconds(_queryEndDate):yyyy-MM-ddTHH:mm:ss.fffZ}");
            Console.WriteLine($"Successfully uploaded {totalEntries} log entries");

            return true;
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "An error occurred while generating sample logs.");
            return false;
        }
    }

    /// <summary>
    /// Executes the large query workflow.
    /// </summary>
```

```
    public static async Task ExecuteLargeQuery()
    {
        Console.WriteLine("Starting recursive query to retrieve all logs...");
        Console.WriteLine();

        var queryLimit = PromptUserForInteger("Enter the query limit (max 10000) ",
10000);
        if (queryLimit > 10000) queryLimit = 10000;

        var queryString = "fields @timestamp, @message | sort @timestamp asc";

        var stopwatch = Stopwatch.StartNew();
        var allResults = await PerformLargeQuery(_logGroupName, queryString,
_queryStartDate, _queryEndDate, queryLimit);
        stopwatch.Stop();

        Console.WriteLine();
        Console.WriteLine($"Queries finished in {stopwatch.Elapsed.TotalSeconds:F3}
seconds.");
        Console.WriteLine($"Total logs found: {allResults.Count}");

        // Check for duplicates
        Console.WriteLine();
        Console.WriteLine("Checking for duplicate logs...");
        var duplicates = FindDuplicateLogs(allResults);
        if (duplicates.Count > 0)
        {
            Console.WriteLine($"WARNING: Found {duplicates.Count} duplicate log
entries!");
            Console.WriteLine("Duplicate entries (showing first 10):");
            foreach (var dup in duplicates.Take(10))
            {
                Console.WriteLine($"  [{dup.Timestamp}] {dup.Message} (appears
{dup.Count} times)");
            }

            var uniqueCount = allResults.Count - duplicates.Sum(d => d.Count - 1);
            Console.WriteLine($"Unique logs: {uniqueCount}");
        }
        else
        {
            Console.WriteLine("No duplicates found. All logs are unique.");
        }
        Console.WriteLine();
```

```
        var viewSample = !_interactive || GetYesNoResponse("Would you like to see a
sample of the logs? (y/n) ");
        if (viewSample)
        {
            Console.WriteLine();
            Console.WriteLine($"Sample logs (first 10 of {allResults.Count}):");
            for (int i = 0; i < Math.Min(10, allResults.Count); i++)
            {
                var timestamp = allResults[i].Find(f => f.Field ==
"@timestamp")?.Value ?? "N/A";
                var message = allResults[i].Find(f => f.Field ==
"@message")?.Value ?? "N/A";
                Console.WriteLine($"[{timestamp}] {message}");
            }
        }
    }

    /// <summary>
    /// Performs a large query using recursive binary search.
    /// </summary>
    private static async Task<List<List<ResultField>>> PerformLargeQuery(
        string logGroupName,
        string queryString,
        long startTime,
        long endTime,
        int limit)
    {
        var queryId = await _wrapper.StartQueryAsync(logGroupName, queryString,
startTime, endTime, limit);
        if (queryId == null)
        {
            return new List<List<ResultField>>();
        }

        var results = await PollQueryResults(queryId);
        if (results == null || results.Count == 0)
        {
            return new List<List<ResultField>>();
        }

        var startDate =
DateTimeOffset.FromUnixTimeSeconds(startTime).ToString("yyyy-MM-ddTHH:mm:ss.fffZ");
```

```
        var endDate = DateTimeOffset.FromUnixTimeSeconds(endTime).ToString("yyyy-MM-
ddTHH:mm:ss.fffZ");
        Console.WriteLine($"Query date range: {startDate} ({startTime}s) to
 {endDate} ({endTime}s). Found {results.Count} logs.");

        if (results.Count < limit)
        {
            Console.WriteLine($"  -> Returning {results.Count} logs (less than limit
 of {limit})");
            return results;
        }

        Console.WriteLine($"  -> Hit limit of {limit}. Need to split and recurse.");

        // Get the timestamp of the last log (sorted to find the actual last one)
        var lastLogTimestamp = GetLastLogTimestamp(results);
        if (lastLogTimestamp == null)
        {
            Console.WriteLine($"  -> No timestamp found in results. Returning
 {results.Count} logs.");
            return results;
        }

        Console.WriteLine($"  -> Last log timestamp: {lastLogTimestamp}");

        // Parse the timestamp and add 1 millisecond to avoid querying the same log
 again
        var lastLogDate = DateTimeOffset.Parse(lastLogTimestamp + " +0000");
        Console.WriteLine($"  -> Last log as DateTimeOffset: {lastLogDate:yyyy-MM-
ddTHH:mm:ss.fffZ} ({lastLogDate.ToUnixTimeSeconds()}s)");

        var offsetLastLogDate = lastLogDate.AddMilliseconds(1);
        Console.WriteLine($"  -> Offset timestamp (last
 + 1ms): {offsetLastLogDate:yyyy-MM-ddTHH:mm:ss.fffZ}
 ({offsetLastLogDate.ToUnixTimeSeconds()}s)");

        // Convert to seconds, but round UP to the next second to avoid overlapping
 with logs in the same second
        // This ensures we don't re-query logs that share the same second as the
 last log
        var offsetLastLogTime = offsetLastLogDate.ToUnixTimeSeconds();
        if (offsetLastLogDate.Millisecond > 0)
        {
            offsetLastLogTime++; // Move to the next full second
```

```
            Console.WriteLine($"  -> Adjusted to next full second:
 {offsetLastLogTime}s ({DateTimeOffset.FromUnixTimeSeconds(offsetLastLogTime):yyyy-
MM-ddTHH:mm:ss.fffZ})");
        }

        Console.WriteLine($"  -> Comparing: offsetLastLogTime={offsetLastLogTime}s
 vs endTime={endTime}s");
        Console.WriteLine($"  -> End time as date:
 {DateTimeOffset.FromUnixTimeSeconds(endTime):yyyy-MM-ddTHH:mm:ss.fffZ}");

        // Check if there's any time range left to query
        if (offsetLastLogTime >= endTime)
        {
            Console.WriteLine($"  -> No time range left to query. Offset time
 ({offsetLastLogTime}s) >= end time ({endTime}s)");
            return results;
        }

        // Split the remaining date range in half
        var (range1Start, range1End, range2Start, range2End) =
 SplitDateRange(offsetLastLogTime, endTime);

        var range1StartDate =
 DateTimeOffset.FromUnixTimeSeconds(range1Start).ToString("yyyy-MM-
ddTHH:mm:ss.fffZ");
        var range1EndDate =
 DateTimeOffset.FromUnixTimeSeconds(range1End).ToString("yyyy-MM-ddTHH:mm:ss.fffZ");
        var range2StartDate =
 DateTimeOffset.FromUnixTimeSeconds(range2Start).ToString("yyyy-MM-
ddTHH:mm:ss.fffZ");
        var range2EndDate =
 DateTimeOffset.FromUnixTimeSeconds(range2End).ToString("yyyy-MM-ddTHH:mm:ss.fffZ");

        Console.WriteLine($"  -> Splitting remaining range:");
        Console.WriteLine($"     Range 1: {range1StartDate} ({range1Start}s) to
 {range1EndDate} ({range1End}s)");
        Console.WriteLine($"     Range 2: {range2StartDate} ({range2Start}s) to
 {range2EndDate} ({range2End}s)");

        // Query both halves recursively
        Console.WriteLine($"  -> Querying range 1...");
        var results1 = await PerformLargeQuery(logGroupName, queryString,
 range1Start, range1End, limit);
        Console.WriteLine($"  -> Range 1 returned {results1.Count} logs");
```

```
        Console.WriteLine($"  -> Querying range 2...");
        var results2 = await PerformLargeQuery(logGroupName, queryString,
range2Start, range2End, limit);
        Console.WriteLine($"  -> Range 2 returned {results2.Count} logs");

        // Combine all results
        var allResults = new List<List<ResultField>>(results);
        allResults.AddRange(results1);
        allResults.AddRange(results2);

        Console.WriteLine($"  -> Combined total: {allResults.Count} logs
({results.Count} + {results1.Count} + {results2.Count})");

        return allResults;
    }

    /// <summary>
    /// Gets the timestamp string of the most recent log from a list of logs.
    /// Sorts timestamps to find the actual last one.
    /// </summary>
    private static string? GetLastLogTimestamp(List<List<ResultField>> logs)
    {
        var timestamps = logs
            .Select(log => log.Find(f => f.Field == "@timestamp")?.Value)
            .Where(t => !string.IsNullOrEmpty(t))
            .OrderBy(t => t)
            .ToList();

        if (timestamps.Count == 0)
        {
            return null;
        }

        return timestamps[timestamps.Count - 1];
    }

    /// <summary>
    /// Splits a date range in half.
    /// Range 2 starts at midpoint + 1 second to avoid overlap.
    /// </summary>
    private static (long range1Start, long range1End, long range2Start, long
range2End) SplitDateRange(long startTime, long endTime)
    {
```

```
        var midpoint = startTime + (endTime - startTime) / 2;
        // Range 2 starts at midpoint + 1 to avoid querying the same second twice
        return (startTime, midpoint, midpoint + 1, endTime);
    }

    /// <summary>
    /// Polls for query results until complete.
    /// </summary>
    private static async Task<List<List<ResultField>>?> PollQueryResults(string
queryId)
    {
        int retryCount = 0;
        const int maxRetries = 60;
        const int retryDelay = 1000;

        while (retryCount < maxRetries)
        {
            var response = await _wrapper.GetQueryResultsAsync(queryId);
            if (response == null)
            {
                return null;
            }

            if (response.Status == QueryStatus.Complete)
            {
                return response.Results;
            }

            if (response.Status == QueryStatus.Failed ||
                response.Status == QueryStatus.Cancelled ||
                response.Status == QueryStatus.Timeout ||
                response.Status == QueryStatus.Unknown)
            {
                _logger.LogError($"Query failed with status: {response.Status}");
                return null;
            }

            await Task.Delay(retryDelay);
            retryCount++;
        }

        _logger.LogError("Timed out waiting for query results.");
        return null;
    }
```

```
   /// <summary>
   /// Cleans up the resources created during the scenario.
   /// </summary>
   public static async Task<bool> Cleanup()
   {
       var cleanup = !_interactive || GetYesNoResponse(
           "Do you want to delete the CloudFormation stack and all resources? (y/n)
");

       if (cleanup)
       {
           try
           {
               var stackDeleteSuccess = await DeleteCloudFormationStack(_stackName,
false);
               return stackDeleteSuccess;
           }
           catch (Exception ex)
           {
               _logger.LogError(ex, "An error occurred while cleaning up the
resources.");
               return false;
           }
       }

       Console.WriteLine($"Resources will remain. Stack name: {_stackName}, Log
group: {_logGroupName}");
       _logger.LogInformation("CloudWatch Logs Large Query scenario is complete.");
       return true;
   }

   /// <summary>
   /// Deletes the CloudFormation stack and waits for confirmation.
   /// </summary>
   private static async Task<bool> DeleteCloudFormationStack(string stackName, bool
forceDelete)
   {
       var request = new DeleteStackRequest
       {
           StackName = stackName,
       };

       if (forceDelete)
```

```
        {
            request.DeletionMode = DeletionMode.FORCE_DELETE_STACK;
        }

        await _amazonCloudFormation.DeleteStackAsync(request);
        Console.WriteLine($"CloudFormation stack '{stackName}' is being deleted.
This may take a few minutes.");

        bool stackDeleted = await WaitForStackDeletion(stackName, forceDelete);

        if (stackDeleted)
        {
            Console.WriteLine($"CloudFormation stack '{stackName}' has been
deleted.");
            return true;
        }
        else
        {
            _logger.LogError($"Failed to delete CloudFormation stack
'{stackName}'.");
            return false;
        }
    }

    /// <summary>
    /// Waits for the stack to be deleted.
    /// </summary>
    private static async Task<bool> WaitForStackDeletion(string stackName, bool
forceDelete)
    {
        int retryCount = 0;
        const int maxRetries = 30;
        const int retryDelay = 10000;

        while (retryCount < maxRetries)
        {
            var describeStacksRequest = new DescribeStacksRequest
            {
                StackName = stackName
            };

            try
            {
```

```
                var describeStacksResponse = await
_amazonCloudFormation.DescribeStacksAsync(describeStacksRequest);

                if (describeStacksResponse.Stacks.Count == 0 ||
                    describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.DELETE_COMPLETE)
                {
                    return true;
                }

                if (!forceDelete && describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.DELETE_FAILED)
                {
                    return await DeleteCloudFormationStack(stackName, true);
                }
            }
            catch (AmazonCloudFormationException ex) when (ex.ErrorCode ==
"ValidationError")
            {
                return true;
            }

            Console.WriteLine($"Waiting for CloudFormation stack '{stackName}' to be
deleted...");
            await Task.Delay(retryDelay);
            retryCount++;
        }

        _logger.LogError($"Timed out waiting for CloudFormation stack '{stackName}'
to be deleted.");
        return false;
    }

    /// <summary>
    /// Waits with a countdown display.
    /// </summary>
    private static async Task WaitWithCountdown(int seconds)
    {
        for (int i = seconds; i > 0; i--)
        {
            Console.Write($"\rWaiting: {i} seconds remaining...  ");
            await Task.Delay(1000);
        }
        Console.WriteLine("\rWait complete.                      ");
```

```
    }

    /// <summary>
    /// Helper method to get a yes or no response from the user.
    /// </summary>
    private static bool GetYesNoResponse(string question)
    {
        Console.WriteLine(question);
        var ynResponse = Console.ReadLine();
        var response = ynResponse != null && ynResponse.Equals("y",
StringComparison.InvariantCultureIgnoreCase);
        return response;
    }

    /// <summary>
    /// Prompts the user for a stack name.
    /// </summary>
    private static string PromptUserForStackName()
    {
        if (_interactive)
        {
            Console.Write($"Enter a name for the CloudFormation stack (press Enter
for default '{_stackName}'): ");
            string? input = Console.ReadLine();
            if (!string.IsNullOrWhiteSpace(input))
            {
                var regex = "[a-zA-Z][-a-zA-Z0-9]*";
                if (!Regex.IsMatch(input, regex))
                {
                    Console.WriteLine($"Invalid stack name. Using default:
{_stackName}");
                    return _stackName;
                }
                return input;
            }
        }
        return _stackName;
    }

    /// <summary>
    /// Prompts the user for input with a default value.
    /// </summary>
    private static string PromptUserForInput(string prompt, string defaultValue)
    {
```

```
        if (_interactive)
        {
            Console.Write($"{prompt}(press Enter for default '{defaultValue}'): ");
            string? input = Console.ReadLine();
            return string.IsNullOrWhiteSpace(input) ? defaultValue : input;
        }
        return defaultValue;
    }

    /// <summary>
    /// Prompts the user for an integer value.
    /// </summary>
    private static int PromptUserForInteger(string prompt, int defaultValue)
    {
        if (_interactive)
        {
            Console.Write($"{prompt}(press Enter for default '{defaultValue}'): ");
            string? input = Console.ReadLine();
            if (string.IsNullOrWhiteSpace(input) || !int.TryParse(input, out var
result))
            {
                return defaultValue;
            }
            return result;
        }
        return defaultValue;
    }

    /// <summary>
    /// Prompts the user for a long value.
    /// </summary>
    private static long PromptUserForLong(string prompt)
    {
        if (_interactive)
        {
            Console.Write(prompt);
            string? input = Console.ReadLine();
            if (long.TryParse(input, out var result))
            {
                return result;
            }
        }
        return 0;
    }
```

```
    /// <summary>
    /// Finds duplicate log entries based on timestamp and message.
    /// </summary>
    private static List<(string Timestamp, string Message, int Count)>
 FindDuplicateLogs(List<List<ResultField>> logs)
    {
        var logSignatures = new Dictionary<string, int>();

        foreach (var log in logs)
        {
            var timestamp = log.Find(f => f.Field == "@timestamp")?.Value ?? "";
            var message = log.Find(f => f.Field == "@message")?.Value ?? "";
            var signature = $"{timestamp}|{message}";

            if (logSignatures.ContainsKey(signature))
            {
                logSignatures[signature]++;
            }
            else
            {
                logSignatures[signature] = 1;
            }
        }

        return logSignatures
            .Where(kvp => kvp.Value > 1)
            .Select(kvp =>
            {
                var parts = kvp.Key.Split('|');
                return (Timestamp: parts[0], Message: parts[1], Count: kvp.Value);
            })
            .OrderByDescending(x => x.Count)
            .ToList();
    }
}
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para .NET .
  - GetQueryResults
  - StartQuery

Exemplos de provedores de identidade Amazon Cognito usando SDK para .NET (v4)

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET (v4) com o Amazon Cognito Identity Provider.

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar perfis de serviço individuais, você pode ver as ações no contexto em seus cenários relacionados.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- Ações

Ações

`ListUserPools`

O código de exemplo a seguir mostra como usar `ListUserPools`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
/// <summary>
/// List the Amazon Cognito user pools for an account.
/// </summary>
/// <returns>A list of UserPoolDescriptionType objects.</returns>
public async Task<List<UserPoolDescriptionType>> ListUserPoolsAsync()
{
    var userPools = new List<UserPoolDescriptionType>();
```

```
        var userPoolsPaginator = _cognitoService.Paginators.ListUserPools(new
ListUserPoolsRequest());

        await foreach (var response in userPoolsPaginator.Responses)
        {
            userPools.AddRange(response.UserPools);
        }

        return userPools;
    }
```

- Para obter detalhes da API, consulte ListUserPoolsa Referência AWS SDK para .NET da API.

AWS Control Tower exemplos usando SDK para .NET (v4)

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET (v4) com AWS Control Tower.

As noções básicas são exemplos de código que mostram como realizar as operações essenciais em um serviço.

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar perfis de serviço individuais, você pode ver as ações no contexto em seus cenários relacionados.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- Conceitos básicos
- Conceitos básicos
- Ações

Conceitos básicos

Olá AWS Control Tower

O exemplo de código a seguir mostra como começar a usar o AWS Control Tower.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
using Amazon.ControlTower;
using Amazon.ControlTower.Model;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Logging.Console;
using Microsoft.Extensions.Logging.Debug;
using LogLevel = Microsoft.Extensions.Logging.LogLevel;

namespace ControlTowerActions;

/// <summary>
/// A class that introduces the AWS Control Tower by listing the
/// available baselines for the account.
/// </summary>
public class HelloControlTower
{
    private static ILogger logger = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for AWS Control Tower.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
 LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonControlTower>()
            )
            .Build();
```

```
        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<HelloControlTower>();

        var amazonClient = host.Services.GetRequiredService<IAmazonControlTower>();

        Console.Clear();
        Console.WriteLine("Hello, AWS Control Tower! Let's list available
 baselines:");
        Console.WriteLine();

        var baselines = new List<BaselineSummary>();

        try
        {
            var baselinesPaginator = amazonClient.Paginators.ListBaselines(new
 ListBaselinesRequest());

            await foreach (var response in baselinesPaginator.Responses)
            {
                baselines.AddRange(response.Baselines);
            }

            Console.WriteLine($"{baselines.Count} baseline(s) retrieved.");
            foreach (var baseline in baselines)
            {
                Console.WriteLine($"\t{baseline.Name}");
            }
        }
        catch (Amazon.ControlTower.Model.AccessDeniedException)
        {
            Console.WriteLine("Access denied. Please ensure you have the necessary
 permissions.");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred: {ex.Message}");
        }
    }
}
```

- Para obter detalhes da API, consulte ListBaselinesa Referência AWS SDK para .NET da API.

Conceitos básicos

Conheça os conceitos básicos

O exemplo de código a seguir mostra como:

- Liste as zonas de pouso.
- Liste, ative, obtenha, redefina e desative as linhas de base.
- Liste, ative, obtenha e desative controles.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Execute um cenário interativo demonstrando AWS Control Tower recursos.

```
using Amazon.ControlCatalog;
using Amazon.ControlTower;
using Amazon.ControlTower.Model;
using Amazon.Organizations;
using Amazon.Organizations.Model;
using Amazon.SecurityToken;
using Amazon.SecurityToken.Model;
using ControlTowerActions;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;

namespace ControlTowerBasics;

/// <summary>
/// Scenario class for AWS Control Tower basics.
/// </summary>
public class ControlTowerBasics
{
    public static bool isInteractive = true;
```

```
    public static ILogger logger = null!;
    public static IAmazonOrganizations? orgClient = null;
    public static IAmazonSecurityTokenService? stsClient = null;
    public static ControlTowerWrapper? wrapper = null;
    private static string? ouArn;
    private static bool useLandingZone = false;

    /// <summary>
    /// Main entry point for the AWS Control Tower basics scenario.
    /// </summary>
    /// <param name="args">Command line arguments.</param>
    public static async Task Main(string[] args)
    {
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonControlTower>()
                .AddAWSService<IAmazonControlCatalog>()
                .AddAWSService<IAmazonOrganizations>()
                .AddAWSService<IAmazonSecurityTokenService>()
                .AddTransient<ControlTowerWrapper>()
            )
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<ControlTowerBasics>();

        wrapper = host.Services.GetRequiredService<ControlTowerWrapper>();
        orgClient = host.Services.GetRequiredService<IAmazonOrganizations>();
        stsClient = host.Services.GetRequiredService<IAmazonSecurityTokenService>();

        await RunScenario();
    }

    /// <summary>
    /// Runs the example scenario.
    /// </summary>
    public static async Task RunScenario()
    {
        Console.WriteLine(new string('-', 88));
        Console.WriteLine("\tWelcome to the AWS Control Tower with ControlCatalog
example scenario.");
        Console.WriteLine(new string('-', 88));
        Console.WriteLine("This demo will walk you through working with AWS Control
Tower for landing zones,");
```

```
        Console.WriteLine("managing baselines, and working with controls.");

        try
        {
            var accountId = (await stsClient!.GetCallerIdentityAsync(new
 GetCallerIdentityRequest())).Account;
            Console.WriteLine($"\nAccount ID: {accountId}");

            Console.WriteLine("\nSome demo operations require the use of a landing
 zone.");
            Console.WriteLine("You can use an existing landing zone or opt out of
 these operations in the demo.");
            Console.WriteLine("For instructions on how to set up a landing zone,");
            Console.WriteLine("see https://docs.aws.amazon.com/controltower/latest/
userguide/getting-started-from-console.html");

            // List available landing zones
            var landingZones = await wrapper!.ListLandingZonesAsync();
            if (landingZones.Count > 0)
            {
                Console.WriteLine("\nAvailable Landing Zones:");
                for (int i = 0; i < landingZones.Count; i++)
                {
                    Console.WriteLine($"{i + 1}. {landingZones[i].Arn}");
                }

                Console.Write($"\nDo you want to use the first landing zone in the
 list ({landingZones[0].Arn})? (y/n): ");
                if (GetUserConfirmation())
                {
                    useLandingZone = true;
                    Console.WriteLine($"Using landing zone: {landingZones[0].Arn}");
                    ouArn = await SetupOrganizationAsync();
                }
            }

            // Managing Baselines
            Console.WriteLine("\nManaging Baselines:");
            var baselines = await wrapper.ListBaselinesAsync();
            Console.WriteLine("\nListing available Baselines:");
            BaselineSummary? controlTowerBaseline = null;
            foreach (var baseline in baselines)
            {
                if (baseline.Name == "AWSControlTowerBaseline")
```

```
                    controlTowerBaseline = baseline;
                Console.WriteLine($"  - {baseline.Name}");
            }

            EnabledBaselineSummary? identityCenterBaseline = null;
            string? baselineArn = null;

            if (useLandingZone && ouArn != null)
            {
                Console.WriteLine("\nListing enabled baselines:");
                var enabledBaselines = await wrapper.ListEnabledBaselinesAsync();
                foreach (var baseline in enabledBaselines)
                {
                    if (baseline.BaselineIdentifier.Contains("baseline/
LN25R72TTG6IGPTQ"))
                        identityCenterBaseline = baseline;
                    Console.WriteLine($"  - {baseline.BaselineIdentifier}");
                }

                if (controlTowerBaseline != null)
                {
                    Console.Write("\nDo you want to enable the Control Tower
 Baseline? (y/n): ");
                    if (GetUserConfirmation())
                    {
                        Console.WriteLine("\nEnabling Control Tower Baseline.");
                        var icBaselineArn = identityCenterBaseline?.Arn;
                        baselineArn = await wrapper.EnableBaselineAsync(ouArn,
                            controlTowerBaseline.Arn, "5.0", icBaselineArn ?? "");
                        var alreadyEnabled = false;
                        if (baselineArn != null)
                        {
                            Console.WriteLine($"Enabled baseline ARN:
 {baselineArn}");
                        }
                        else
                        {
                            // Find the enabled baseline
                            foreach (var enabled in enabledBaselines)
                            {
                                if (enabled.BaselineIdentifier ==
 controlTowerBaseline.Arn)
                                {
                                    baselineArn = enabled.Arn;
```

```
                                    alreadyEnabled = true;
                                    Console.WriteLine("No change, the selected
baseline was already enabled.");
                                    break;
                                }
                            }
                        }

                        if (baselineArn != null)
                        {
                            Console.Write("\nDo you want to reset the Control Tower
Baseline? (y/n): ");
                            if (GetUserConfirmation())
                            {
                                Console.WriteLine($"\nResetting Control Tower
Baseline: {baselineArn}");
                                var operationId = await
wrapper.ResetEnabledBaselineAsync(baselineArn);
                                Console.WriteLine($"Reset baseline operation id:
{operationId}");
                            }

                            Console.Write("\nDo you want to disable the Control
Tower Baseline? (y/n): ");
                            if (GetUserConfirmation())
                            {
                                Console.WriteLine($"Disabling baseline ARN:
{baselineArn}");
                                var operationId = await
wrapper.DisableBaselineAsync(baselineArn);
                                Console.WriteLine($"Disabled baseline operation id:
{operationId}");
                                if (alreadyEnabled)
                                {
                                    Console.WriteLine($"\nRe-enabling Control Tower
Baseline: {baselineArn}");
                                    // Re-enable the Control Tower baseline if it
was originally enabled.
                                    await wrapper.EnableBaselineAsync(ouArn,
                                        controlTowerBaseline.Arn, "5.0",
icBaselineArn ?? "");
                                }
                            }
                        }
```

```
                    }
                }
            }

            // Managing Controls
            Console.WriteLine("\nManaging Controls:");
            var controls = await wrapper.ListControlsAsync();
            Console.WriteLine("\nListing first 5 available Controls:");
            for (int i = 0; i < Math.Min(5, controls.Count); i++)
            {
                Console.WriteLine($"{i + 1}. {controls[i].Name} -
 {controls[i].Arn}");
            }

            if (useLandingZone && ouArn != null)
            {
                var enabledControls = await wrapper.ListEnabledControlsAsync(ouArn);
                Console.WriteLine("\nListing enabled controls:");
                for (int i = 0; i < enabledControls.Count; i++)
                {
                    Console.WriteLine($"{i + 1}.
 {enabledControls[i].ControlIdentifier}");
                }

                // Find first non-enabled control
                var enabledControlArns = enabledControls.Select(c =>
 c.Arn).ToHashSet();
                var controlArn = controls.FirstOrDefault(c => !
enabledControlArns.Contains(c.Arn))?.Arn;

                if (controlArn != null)
                {
                    Console.Write($"\nDo you want to enable the control
 {controlArn}? (y/n): ");
                    if (GetUserConfirmation())
                    {
                        Console.WriteLine($"\nEnabling control: {controlArn}");
                        var operationId = await
 wrapper.EnableControlAsync(controlArn, ouArn);
                        if (operationId != null)
                        {
                            Console.WriteLine($"Enabled control with operation id:
 {operationId}");
```

```
                            Console.Write("\nDo you want to disable the control? (y/
n): ");
                            if (GetUserConfirmation())
                            {
                                Console.WriteLine("\nDisabling the control...");
                                var disableOpId = await
 wrapper.DisableControlAsync(controlArn, ouArn);
                                Console.WriteLine($"Disable operation ID:
 {disableOpId}");
                            }
                        }
                    }
                }
            }

            Console.WriteLine("\nThis concludes the example scenario.");
            Console.WriteLine("Thanks for watching!");
            Console.WriteLine(new string('-', 88));
        }
        catch (Exception ex)
        {
            logger.LogError(ex, "An error occurred during the Control Tower
 scenario.");
            Console.WriteLine($"An error occurred: {ex.Message}");
        }
    }

    /// <summary>
    /// Sets up AWS Organizations and creates or finds a Sandbox OU.
    /// </summary>
    /// <returns>The ARN of the Sandbox organizational unit.</returns>
    private static async Task<string> SetupOrganizationAsync()
    {
        Console.WriteLine("\nChecking organization status...");

        try
        {
            var orgResponse = await orgClient!.DescribeOrganizationAsync(new
 DescribeOrganizationRequest());
            var orgId = orgResponse.Organization.Id;
            Console.WriteLine($"Account is part of organization: {orgId}");
        }
        catch (AWSOrganizationsNotInUseException)
        {
```

```
            Console.WriteLine("No organization found. Creating a new
organization...");
            var createResponse = await orgClient!.CreateOrganizationAsync(new
CreateOrganizationRequest { FeatureSet = OrganizationFeatureSet.ALL });
            var orgId = createResponse.Organization.Id;
            Console.WriteLine($"Created new organization: {orgId}");
        }

        // Look for Sandbox OU
        var roots = await orgClient.ListRootsAsync(new ListRootsRequest());
        var rootId = roots.Roots[0].Id;

        Console.WriteLine("Checking for Sandbox OU...");
        var ous = await orgClient.ListOrganizationalUnitsForParentAsync(new
ListOrganizationalUnitsForParentRequest { ParentId = rootId });
        var sandboxOu = ous.OrganizationalUnits.FirstOrDefault(ou => ou.Name ==
"Sandbox");

        if (sandboxOu == null)
        {
            Console.WriteLine("Creating Sandbox OU...");
            var createOuResponse = await orgClient.CreateOrganizationalUnitAsync(new
CreateOrganizationalUnitRequest { ParentId = rootId, Name = "Sandbox" });
            sandboxOu = createOuResponse.OrganizationalUnit;
            Console.WriteLine($"Created new Sandbox OU: {sandboxOu.Id}");
        }
        else
        {
            Console.WriteLine($"Found existing Sandbox OU: {sandboxOu.Id}");
        }

        return sandboxOu.Arn;
    }

    /// <summary>
    /// Gets user confirmation by waiting for input or returning true if not
interactive.
    /// </summary>
    /// <returns>True if user enters 'y' or if isInteractive is false, otherwise
false.</returns>
    private static bool GetUserConfirmation()
    {
        return Console.ReadLine()?.ToLower() == "y" || !isInteractive;
    }
```

```
}
```

Métodos de encapsulamento que são chamados pelo cenário para gerenciar as ações do Aurora.

```
using Amazon.ControlCatalog;
using Amazon.ControlCatalog.Model;
using Amazon.ControlTower;
using Amazon.ControlTower.Model;
using ValidationException = Amazon.ControlTower.Model.ValidationException;

namespace ControlTowerActions;

/// <summary>
/// Methods to perform AWS Control Tower actions.
/// </summary>
public class ControlTowerWrapper
{
    private readonly IAmazonControlTower _controlTowerService;
    private readonly IAmazonControlCatalog _controlCatalogService;

    /// <summary>
    /// Constructor for the wrapper class containing AWS Control Tower actions.
    /// </summary>
    /// <param name="controlTowerService">The AWS Control Tower client object.</
param>
    /// <param name="controlCatalogService">The AWS Control Catalog client object.</
param>
    public ControlTowerWrapper(IAmazonControlTower controlTowerService,
 IAmazonControlCatalog controlCatalogService)
    {
        _controlTowerService = controlTowerService;
        _controlCatalogService = controlCatalogService;
    }

    /// <summary>
    /// List the AWS Control Tower landing zones for an account.
    /// </summary>
    /// <returns>A list of LandingZoneSummary objects.</returns>
    public async Task<List<LandingZoneSummary>> ListLandingZonesAsync()
    {
```

```
        try
        {
            var landingZones = new List<LandingZoneSummary>();

            var landingZonesPaginator =
_controlTowerService.Paginators.ListLandingZones(new ListLandingZonesRequest());

            await foreach (var response in landingZonesPaginator.Responses)
            {
                landingZones.AddRange(response.LandingZones);
            }

            return landingZones;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't list landing zones. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// List all baselines.
    /// </summary>
    /// <returns>A list of baseline summaries.</returns>
    public async Task<List<BaselineSummary>> ListBaselinesAsync()
    {
        try
        {
            var baselines = new List<BaselineSummary>();

            var baselinesPaginator =
_controlTowerService.Paginators.ListBaselines(new ListBaselinesRequest());

            await foreach (var response in baselinesPaginator.Responses)
            {
                baselines.AddRange(response.Baselines);
            }

            return baselines;
        }
        catch (AmazonControlTowerException ex)
```

```
        {
            Console.WriteLine($"Couldn't list baselines. Here's why: {ex.ErrorCode}:
 {ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// List all enabled baselines.
    /// </summary>
    /// <returns>A list of enabled baseline summaries.</returns>
    public async Task<List<EnabledBaselineSummary>> ListEnabledBaselinesAsync()
    {
        try
        {
            var enabledBaselines = new List<EnabledBaselineSummary>();

            var enabledBaselinesPaginator =
 _controlTowerService.Paginators.ListEnabledBaselines(new
 ListEnabledBaselinesRequest());

            await foreach (var response in enabledBaselinesPaginator.Responses)
            {
                enabledBaselines.AddRange(response.EnabledBaselines);
            }

            return enabledBaselines;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't list enabled baselines. Here's why:
 {ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// Enable a baseline for the specified target.
    /// </summary>
    /// <param name="targetIdentifier">The ARN of the target.</param>
    /// <param name="baselineIdentifier">The identifier of baseline to enable.</
param>
```

```
   /// <param name="baselineVersion">The version of baseline to enable.</param>
   /// <param name="identityCenterBaseline">The identifier of identity center
baseline if it is enabled.</param>
   /// <returns>The enabled baseline ARN or null.</returns>
   public async Task<string?> EnableBaselineAsync(string targetIdentifier, string
baselineIdentifier, string baselineVersion, string identityCenterBaseline)
   {
       try
       {
           var parameters = new List<EnabledBaselineParameter>();
           if (!string.IsNullOrEmpty(identityCenterBaseline))
           {
               parameters.Add(
                   new EnabledBaselineParameter
                   {
                       Key = "IdentityCenterEnabledBaselineArn",
                       Value = identityCenterBaseline
                   });
           }
           var request = new EnableBaselineRequest
           {
               BaselineIdentifier = baselineIdentifier,
               BaselineVersion = baselineVersion,
               TargetIdentifier = targetIdentifier,
               Parameters = parameters
           };

           var response = await _controlTowerService.EnableBaselineAsync(request);
           var operationId = response.OperationIdentifier;

           // Wait for operation to complete
           while (true)
           {
               var status = await GetBaselineOperationAsync(operationId);
               Console.WriteLine($"Baseline operation status: {status}");
               if (status == BaselineOperationStatus.SUCCEEDED || status ==
BaselineOperationStatus.FAILED)
               {
                   break;
               }
               await Task.Delay(30000); // Wait 30 seconds
           }

           return response.Arn;
```

```
        }
        catch (ValidationException ex)
        {
            if (ex.Message.Contains("already enabled"))
                Console.WriteLine("Baseline is already enabled for this target");
            else { Console.WriteLine(ex.Message); }
            // Write the message and return null if baseline cannot be enabled.
            return null;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't enable baseline. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// Disable a baseline for a specific target and wait for the operation to
complete.
    /// </summary>
    /// <param name="enabledBaselineIdentifier">The identifier of the baseline to
disable.</param>
    /// <returns>The operation ID or null if there was a conflict.</returns>
    public async Task<string?> DisableBaselineAsync(string
enabledBaselineIdentifier)
    {
        try
        {
            var request = new DisableBaselineRequest
            {
                EnabledBaselineIdentifier = enabledBaselineIdentifier
            };

            var response = await _controlTowerService.DisableBaselineAsync(request);
            var operationId = response.OperationIdentifier;

            // Wait for operation to complete
            while (true)
            {
                var status = await GetBaselineOperationAsync(operationId);
                Console.WriteLine($"Baseline operation status: {status}");
```

```
                if (status == BaselineOperationStatus.SUCCEEDED || status ==
BaselineOperationStatus.FAILED)
                {
                    break;
                }
                await Task.Delay(30000); // Wait 30 seconds
            }

            return operationId;
        }
        catch (ConflictException ex)
        {
            Console.WriteLine($"Conflict disabling baseline: {ex.Message}. Skipping
disable step.");
            return null;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't disable baseline. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// Reset an enabled baseline for a specific target.
    /// </summary>
    /// <param name="enabledBaselineIdentifier">The identifier of the enabled
baseline to reset.</param>
    /// <returns>The operation ID.</returns>
    public async Task<string> ResetEnabledBaselineAsync(string
enabledBaselineIdentifier)
    {
        try
        {
            var request = new ResetEnabledBaselineRequest
            {
                EnabledBaselineIdentifier = enabledBaselineIdentifier
            };

            var response = await
_controlTowerService.ResetEnabledBaselineAsync(request);
            var operationId = response.OperationIdentifier;
```

```
            // Wait for operation to complete
            while (true)
            {
                var status = await GetBaselineOperationAsync(operationId);
                Console.WriteLine($"Baseline operation status: {status}");
                if (status == BaselineOperationStatus.SUCCEEDED || status ==
BaselineOperationStatus.FAILED)
                {
                    break;
                }
                await Task.Delay(30000); // Wait 30 seconds
            }

            return operationId;
        }
        catch (Amazon.ControlTower.Model.ResourceNotFoundException)
        {
            Console.WriteLine("Target not found, unable to reset enabled
baseline.");
            throw;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't reset enabled baseline. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// Get the status of a baseline operation.
    /// </summary>
    /// <param name="operationId">The ID of the baseline operation.</param>
    /// <returns>The operation status.</returns>
    public async Task<BaselineOperationStatus> GetBaselineOperationAsync(string
operationId)
    {
        try
        {
            var request = new GetBaselineOperationRequest
            {
                OperationIdentifier = operationId
```

```
            };

            var response = await
 _controlTowerService.GetBaselineOperationAsync(request);
            return response.BaselineOperation.Status;
        }
        catch (Amazon.ControlTower.Model.ResourceNotFoundException)
        {
            Console.WriteLine("Operation not found.");
            throw;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't get baseline operation status. Here's why:
 {ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// List enabled controls for a target organizational unit.
    /// </summary>
    /// <param name="targetIdentifier">The target organizational unit identifier.</
param>
    /// <returns>A list of enabled control summaries.</returns>
    public async Task<List<EnabledControlSummary>> ListEnabledControlsAsync(string
 targetIdentifier)
    {
        try
        {
            var request = new ListEnabledControlsRequest
            {
                TargetIdentifier = targetIdentifier
            };

            var enabledControls = new List<EnabledControlSummary>();

            var enabledControlsPaginator =
 _controlTowerService.Paginators.ListEnabledControls(request);

            await foreach (var response in enabledControlsPaginator.Responses)
            {
                enabledControls.AddRange(response.EnabledControls);
```

```
            }

            return enabledControls;
        }
        catch (Amazon.ControlTower.Model.ResourceNotFoundException ex) when
(ex.Message.Contains("not registered with AWS Control Tower"))
        {
            Console.WriteLine("AWS Control Tower must be enabled to work with
enabling controls.");
            return new List<EnabledControlSummary>();
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't list enabled controls. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// Enable a control for a specified target.
    /// </summary>
    /// <param name="controlArn">The ARN of the control to enable.</param>
    /// <param name="targetIdentifier">The identifier of the target (e.g., OU
ARN).</param>
    /// <returns>The operation ID or null if already enabled.</returns>
    public async Task<string?> EnableControlAsync(string controlArn, string
targetIdentifier)
    {
        try
        {
            Console.WriteLine(controlArn);
            Console.WriteLine(targetIdentifier);

            var request = new EnableControlRequest
            {
                ControlIdentifier = controlArn,
                TargetIdentifier = targetIdentifier
            };

            var response = await _controlTowerService.EnableControlAsync(request);
            var operationId = response.OperationIdentifier;
```

```
            // Wait for operation to complete
            while (true)
            {
                var status = await GetControlOperationAsync(operationId);
                Console.WriteLine($"Control operation status: {status}");
                if (status == ControlOperationStatus.SUCCEEDED || status ==
ControlOperationStatus.FAILED)
                {
                    break;
                }
                await Task.Delay(30000); // Wait 30 seconds
            }

            return operationId;
        }
        catch (Amazon.ControlTower.Model.ValidationException ex) when
(ex.Message.Contains("already enabled"))
        {
            Console.WriteLine("Control is already enabled for this target");
            return null;
        }
        catch (Amazon.ControlTower.Model.ResourceNotFoundException ex) when
(ex.Message.Contains("not registered with AWS Control Tower"))
        {
            Console.WriteLine("AWS Control Tower must be enabled to work with
enabling controls.");
            return null;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't enable control. Here's why: {ex.ErrorCode}:
{ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// Disable a control for a specified target.
    /// </summary>
    /// <param name="controlArn">The ARN of the control to disable.</param>
    /// <param name="targetIdentifier">The identifier of the target (e.g., OU
ARN).</param>
    /// <returns>The operation ID.</returns>
```

```
   public async Task<string> DisableControlAsync(string controlArn, string
targetIdentifier)
   {
       try
       {
           var request = new DisableControlRequest
           {
               ControlIdentifier = controlArn,
               TargetIdentifier = targetIdentifier
           };

           var response = await _controlTowerService.DisableControlAsync(request);
           var operationId = response.OperationIdentifier;

           // Wait for operation to complete
           while (true)
           {
               var status = await GetControlOperationAsync(operationId);
               Console.WriteLine($"Control operation status: {status}");
               if (status == ControlOperationStatus.SUCCEEDED || status ==
ControlOperationStatus.FAILED)
               {
                   break;
               }
               await Task.Delay(30000); // Wait 30 seconds
           }

           return operationId;
       }
       catch (Amazon.ControlTower.Model.ResourceNotFoundException)
       {
           Console.WriteLine("Control not found.");
           throw;
       }
       catch (AmazonControlTowerException ex)
       {
           Console.WriteLine($"Couldn't disable control. Here's why:
{ex.ErrorCode}: {ex.Message}");
           throw;
       }
   }


   /// <summary>
```

```
    /// Get the status of a control operation.
    /// </summary>
    /// <param name="operationId">The ID of the control operation.</param>
    /// <returns>The operation status.</returns>
    public async Task<ControlOperationStatus> GetControlOperationAsync(string
operationId)
    {
        try
        {
            var request = new GetControlOperationRequest
            {
                OperationIdentifier = operationId
            };

            var response = await
_controlTowerService.GetControlOperationAsync(request);
            return response.ControlOperation.Status;
        }
        catch (Amazon.ControlTower.Model.ResourceNotFoundException)
        {
            Console.WriteLine("Operation not found.");
            throw;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't get control operation status. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }


    /// <summary>
    /// List all controls in the Control Tower control catalog.
    /// </summary>
    /// <returns>A list of control summaries.</returns>
    public async Task<List<ControlSummary>> ListControlsAsync()
    {
        try
        {
            var controls = new List<ControlSummary>();
```

```
            var controlsPaginator =
_controlCatalogService.Paginators.ListControls(new
Amazon.ControlCatalog.Model.ListControlsRequest());

            await foreach (var response in controlsPaginator.Responses)
            {
                controls.AddRange(response.Controls);
            }

            return controls;
        }
        catch (AmazonControlCatalogException ex)
        {
            Console.WriteLine($"Couldn't list controls. Here's why: {ex.ErrorCode}:
{ex.Message}");
            throw;
        }
    }

}
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para .NET .
  - CreateLandingZone
  - DeleteLandingZone
  - DisableBaseline
  - DisableControl
  - EnableBaseline
  - EnableControl
  - GetControlOperation
  - GetLandingZoneOperation
  - ListBaselines
  - ListEnabledBaselines
  - ListEnabledControls
  - ListLandingZones
  - ResetEnabledBaseline

Ações

`DisableBaseline`

O código de exemplo a seguir mostra como usar `DisableBaseline`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
   /// <summary>
   /// Disable a baseline for a specific target and wait for the operation to
complete.
   /// </summary>
   /// <param name="enabledBaselineIdentifier">The identifier of the baseline to
disable.</param>
   /// <returns>The operation ID or null if there was a conflict.</returns>
   public async Task<string?> DisableBaselineAsync(string
enabledBaselineIdentifier)
   {
       try
       {
           var request = new DisableBaselineRequest
           {
               EnabledBaselineIdentifier = enabledBaselineIdentifier
           };

           var response = await _controlTowerService.DisableBaselineAsync(request);
           var operationId = response.OperationIdentifier;

           // Wait for operation to complete
           while (true)
           {
               var status = await GetBaselineOperationAsync(operationId);
               Console.WriteLine($"Baseline operation status: {status}");
               if (status == BaselineOperationStatus.SUCCEEDED || status ==
BaselineOperationStatus.FAILED)
               {
```

```
                    break;
                }
                await Task.Delay(30000); // Wait 30 seconds
            }

            return operationId;
        }
        catch (ConflictException ex)
        {
            Console.WriteLine($"Conflict disabling baseline: {ex.Message}. Skipping
disable step.");
            return null;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't disable baseline. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }
```

- Para obter detalhes da API, consulte DisableBaselinea Referência AWS SDK para .NET da API.

`DisableControl`

O código de exemplo a seguir mostra como usar `DisableControl`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Disable a control for a specified target.
    /// </summary>
```

```
   /// <param name="controlArn">The ARN of the control to disable.</param>
   /// <param name="targetIdentifier">The identifier of the target (e.g., OU
ARN).</param>
   /// <returns>The operation ID.</returns>
   public async Task<string> DisableControlAsync(string controlArn, string
targetIdentifier)
   {
       try
       {
           var request = new DisableControlRequest
           {
               ControlIdentifier = controlArn,
               TargetIdentifier = targetIdentifier
           };

           var response = await _controlTowerService.DisableControlAsync(request);
           var operationId = response.OperationIdentifier;

           // Wait for operation to complete
           while (true)
           {
               var status = await GetControlOperationAsync(operationId);
               Console.WriteLine($"Control operation status: {status}");
               if (status == ControlOperationStatus.SUCCEEDED || status ==
ControlOperationStatus.FAILED)
               {
                   break;
               }
               await Task.Delay(30000); // Wait 30 seconds
           }

           return operationId;
       }
       catch (Amazon.ControlTower.Model.ResourceNotFoundException)
       {
           Console.WriteLine("Control not found.");
           throw;
       }
       catch (AmazonControlTowerException ex)
       {
           Console.WriteLine($"Couldn't disable control. Here's why:
{ex.ErrorCode}: {ex.Message}");
           throw;
       }
```

```
    }
```

- Para obter detalhes da API, consulte DisableControla Referência AWS SDK para .NET da API.

`EnableBaseline`

O código de exemplo a seguir mostra como usar `EnableBaseline`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Enable a baseline for the specified target.
    /// </summary>
    /// <param name="targetIdentifier">The ARN of the target.</param>
    /// <param name="baselineIdentifier">The identifier of baseline to enable.</
param>
    /// <param name="baselineVersion">The version of baseline to enable.</param>
    /// <param name="identityCenterBaseline">The identifier of identity center
 baseline if it is enabled.</param>
    /// <returns>The enabled baseline ARN or null.</returns>
    public async Task<string?> EnableBaselineAsync(string targetIdentifier, string
 baselineIdentifier, string baselineVersion, string identityCenterBaseline)
    {
        try
        {
            var parameters = new List<EnabledBaselineParameter>();
            if (!string.IsNullOrEmpty(identityCenterBaseline))
            {
                parameters.Add(
                    new EnabledBaselineParameter
                    {
                        Key = "IdentityCenterEnabledBaselineArn",
                        Value = identityCenterBaseline
                    });
```

```
            }
            var request = new EnableBaselineRequest
            {
                BaselineIdentifier = baselineIdentifier,
                BaselineVersion = baselineVersion,
                TargetIdentifier = targetIdentifier,
                Parameters = parameters
            };

            var response = await _controlTowerService.EnableBaselineAsync(request);
            var operationId = response.OperationIdentifier;

            // Wait for operation to complete
            while (true)
            {
                var status = await GetBaselineOperationAsync(operationId);
                Console.WriteLine($"Baseline operation status: {status}");
                if (status == BaselineOperationStatus.SUCCEEDED || status ==
BaselineOperationStatus.FAILED)
                {
                    break;
                }
                await Task.Delay(30000); // Wait 30 seconds
            }

            return response.Arn;
        }
        catch (ValidationException ex)
        {
            if (ex.Message.Contains("already enabled"))
                Console.WriteLine("Baseline is already enabled for this target");
            else { Console.WriteLine(ex.Message); }
            // Write the message and return null if baseline cannot be enabled.
            return null;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't enable baseline. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }
```

- Para obter detalhes da API, consulte EnableBaselinea Referência AWS SDK para .NET da API.

`EnableControl`

O código de exemplo a seguir mostra como usar `EnableControl`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
   /// <summary>
   /// Enable a control for a specified target.
   /// </summary>
   /// <param name="controlArn">The ARN of the control to enable.</param>
   /// <param name="targetIdentifier">The identifier of the target (e.g., OU
ARN).</param>
   /// <returns>The operation ID or null if already enabled.</returns>
   public async Task<string?> EnableControlAsync(string controlArn, string
targetIdentifier)
   {
       try
       {
           Console.WriteLine(controlArn);
           Console.WriteLine(targetIdentifier);

           var request = new EnableControlRequest
           {
               ControlIdentifier = controlArn,
               TargetIdentifier = targetIdentifier
           };

           var response = await _controlTowerService.EnableControlAsync(request);
           var operationId = response.OperationIdentifier;

           // Wait for operation to complete
```

```
            while (true)
            {
                var status = await GetControlOperationAsync(operationId);
                Console.WriteLine($"Control operation status: {status}");
                if (status == ControlOperationStatus.SUCCEEDED || status ==
ControlOperationStatus.FAILED)
                {
                    break;
                }
                await Task.Delay(30000); // Wait 30 seconds
            }

            return operationId;
        }
        catch (Amazon.ControlTower.Model.ValidationException ex) when
(ex.Message.Contains("already enabled"))
        {
            Console.WriteLine("Control is already enabled for this target");
            return null;
        }
        catch (Amazon.ControlTower.Model.ResourceNotFoundException ex) when
(ex.Message.Contains("not registered with AWS Control Tower"))
        {
            Console.WriteLine("AWS Control Tower must be enabled to work with
enabling controls.");
            return null;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't enable control. Here's why: {ex.ErrorCode}:
{ex.Message}");
            throw;
        }
    }
```

- Para obter detalhes da API, consulte EnableControla Referência AWS SDK para .NET da API.

`GetBaselineOperation`

O código de exemplo a seguir mostra como usar `GetBaselineOperation`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
   /// <summary>
   /// Get the status of a baseline operation.
   /// </summary>
   /// <param name="operationId">The ID of the baseline operation.</param>
   /// <returns>The operation status.</returns>
   public async Task<BaselineOperationStatus> GetBaselineOperationAsync(string
operationId)
   {
       try
       {
           var request = new GetBaselineOperationRequest
           {
               OperationIdentifier = operationId
           };

           var response = await
_controlTowerService.GetBaselineOperationAsync(request);
           return response.BaselineOperation.Status;
       }
       catch (Amazon.ControlTower.Model.ResourceNotFoundException)
       {
           Console.WriteLine("Operation not found.");
           throw;
       }
       catch (AmazonControlTowerException ex)
       {
           Console.WriteLine($"Couldn't get baseline operation status. Here's why:
{ex.ErrorCode}: {ex.Message}");
           throw;
       }
   }
```

- Para obter detalhes da API, consulte GetBaselineOperationa Referência AWS SDK para .NET da API.

`GetControlOperation`

O código de exemplo a seguir mostra como usar `GetControlOperation`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
   /// <summary>
   /// Get the status of a control operation.
   /// </summary>
   /// <param name="operationId">The ID of the control operation.</param>
   /// <returns>The operation status.</returns>
   public async Task<ControlOperationStatus> GetControlOperationAsync(string
operationId)
   {
       try
       {
           var request = new GetControlOperationRequest
           {
               OperationIdentifier = operationId
           };

           var response = await
_controlTowerService.GetControlOperationAsync(request);
           return response.ControlOperation.Status;
       }
       catch (Amazon.ControlTower.Model.ResourceNotFoundException)
       {
           Console.WriteLine("Operation not found.");
           throw;
       }
       catch (AmazonControlTowerException ex)
       {
```

```
            Console.WriteLine($"Couldn't get control operation status. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }
```

- Para obter detalhes da API, consulte GetControlOperationa Referência AWS SDK para .NET da API.

`ListBaselines`

O código de exemplo a seguir mostra como usar `ListBaselines`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// List all baselines.
    /// </summary>
    /// <returns>A list of baseline summaries.</returns>
    public async Task<List<BaselineSummary>> ListBaselinesAsync()
    {
        try
        {
            var baselines = new List<BaselineSummary>();

            var baselinesPaginator =
_controlTowerService.Paginators.ListBaselines(new ListBaselinesRequest());

            await foreach (var response in baselinesPaginator.Responses)
            {
                baselines.AddRange(response.Baselines);
            }
```

```
            return baselines;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't list baselines. Here's why: {ex.ErrorCode}:
{ex.Message}");
            throw;
        }
    }
```

- Para obter detalhes da API, consulte ListBaselinesa Referência AWS SDK para .NET da API.

`ListEnabledBaselines`

O código de exemplo a seguir mostra como usar `ListEnabledBaselines`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// List all enabled baselines.
    /// </summary>
    /// <returns>A list of enabled baseline summaries.</returns>
    public async Task<List<EnabledBaselineSummary>> ListEnabledBaselinesAsync()
    {
        try
        {
            var enabledBaselines = new List<EnabledBaselineSummary>();

            var enabledBaselinesPaginator =
_controlTowerService.Paginators.ListEnabledBaselines(new
ListEnabledBaselinesRequest());

            await foreach (var response in enabledBaselinesPaginator.Responses)
            {
```

```
                enabledBaselines.AddRange(response.EnabledBaselines);
            }

            return enabledBaselines;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't list enabled baselines. Here's why:
 {ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }
```

- Para obter detalhes da API, consulte ListEnabledBaselinesa Referência AWS SDK para .NET da API.

`ListEnabledControls`

O código de exemplo a seguir mostra como usar `ListEnabledControls`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// List enabled controls for a target organizational unit.
    /// </summary>
    /// <param name="targetIdentifier">The target organizational unit identifier.</
param>
    /// <returns>A list of enabled control summaries.</returns>
    public async Task<List<EnabledControlSummary>> ListEnabledControlsAsync(string
 targetIdentifier)
    {
        try
        {
```

```
            var request = new ListEnabledControlsRequest
            {
                TargetIdentifier = targetIdentifier
            };

            var enabledControls = new List<EnabledControlSummary>();

            var enabledControlsPaginator =
_controlTowerService.Paginators.ListEnabledControls(request);

            await foreach (var response in enabledControlsPaginator.Responses)
            {
                enabledControls.AddRange(response.EnabledControls);
            }

            return enabledControls;
        }
        catch (Amazon.ControlTower.Model.ResourceNotFoundException ex) when
(ex.Message.Contains("not registered with AWS Control Tower"))
        {
            Console.WriteLine("AWS Control Tower must be enabled to work with
enabling controls.");
            return new List<EnabledControlSummary>();
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't list enabled controls. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }
```

- Para obter detalhes da API, consulte ListEnabledControlsa Referência AWS SDK para .NET da API.

`ListLandingZones`

O código de exemplo a seguir mostra como usar `ListLandingZones`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// List the AWS Control Tower landing zones for an account.
    /// </summary>
    /// <returns>A list of LandingZoneSummary objects.</returns>
    public async Task<List<LandingZoneSummary>> ListLandingZonesAsync()
    {
        try
        {
            var landingZones = new List<LandingZoneSummary>();

            var landingZonesPaginator =
_controlTowerService.Paginators.ListLandingZones(new ListLandingZonesRequest());

            await foreach (var response in landingZonesPaginator.Responses)
            {
                landingZones.AddRange(response.LandingZones);
            }

            return landingZones;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't list landing zones. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }
```

- Para obter detalhes da API, consulte ListLandingZonesa Referência AWS SDK para .NET da API.

`ResetEnabledBaseline`

O código de exemplo a seguir mostra como usar `ResetEnabledBaseline`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
   /// <summary>
   /// Reset an enabled baseline for a specific target.
   /// </summary>
   /// <param name="enabledBaselineIdentifier">The identifier of the enabled
baseline to reset.</param>
   /// <returns>The operation ID.</returns>
   public async Task<string> ResetEnabledBaselineAsync(string
enabledBaselineIdentifier)
   {
       try
       {
           var request = new ResetEnabledBaselineRequest
           {
               EnabledBaselineIdentifier = enabledBaselineIdentifier
           };

           var response = await
_controlTowerService.ResetEnabledBaselineAsync(request);
           var operationId = response.OperationIdentifier;

           // Wait for operation to complete
           while (true)
           {
               var status = await GetBaselineOperationAsync(operationId);
               Console.WriteLine($"Baseline operation status: {status}");
               if (status == BaselineOperationStatus.SUCCEEDED || status ==
BaselineOperationStatus.FAILED)
               {
                   break;
               }
               await Task.Delay(30000); // Wait 30 seconds
```

```
            }

            return operationId;
        }
        catch (Amazon.ControlTower.Model.ResourceNotFoundException)
        {
            Console.WriteLine("Target not found, unable to reset enabled
baseline.");
            throw;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't reset enabled baseline. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }
```

- Para obter detalhes da API, consulte ResetEnabledBaselinea Referência AWS SDK para .NET da API.

Exemplos SDK para .NET do DynamoDB usando (v4)

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET (v4) com o DynamoDB.

As noções básicas são exemplos de código que mostram como realizar as operações essenciais em um serviço.

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar perfis de serviço individuais, você pode ver as ações no contexto em seus cenários relacionados.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- Conceitos básicos

- Conceitos básicos
- Ações

Conceitos básicos

Olá, DynamoDB

O exemplo de código a seguir mostra como começar a usar o DynamoDB.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.Model;
using Microsoft.Extensions.DependencyInjection;

namespace DynamoDBActions;

/// <summary>
/// A simple example that demonstrates basic DynamoDB operations.
/// </summary>
public class HelloDynamoDB
{
    /// <summary>
    /// HelloDynamoDB lists the existing DynamoDB tables for the default user.
    /// </summary>
    /// <param name="args">Command line arguments</param>
    /// <returns>Async task.</returns>
    static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon DynamoDB.
        using var host =
 Microsoft.Extensions.Hosting.Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonDynamoDB>()
```

```
            )
            .Build();

        // Now the client is available for injection.
        var dynamoDbClient = host.Services.GetRequiredService<IAmazonDynamoDB>();

        try
        {
            var request = new ListTablesRequest();
            var tableNames = new List<string>();

            var paginatorForTables = dynamoDbClient.Paginators.ListTables(request);

            await foreach (var tableName in paginatorForTables.TableNames)
            {
                tableNames.Add(tableName);
            }

            Console.WriteLine("Welcome to the DynamoDB Hello Service example. " +
                              "\nLet's list your DynamoDB tables:");
            tableNames.ForEach(table =>
            {
                Console.WriteLine($"Table: {table}");
            });
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB service error occurred while
 listing tables. {ex.Message}");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while listing tables.
 {ex.Message}");
        }
    }
}
```

- Para obter detalhes da API, consulte ListTablesa Referência AWS SDK para .NET da API.

Conceitos básicos

Conheça os conceitos básicos

O exemplo de código a seguir mostra como:

- Criar uma tabela que possa conter dados de filmes.
- Colocar, obter e atualizar um único filme na tabela.
- Gravar dados de filmes na tabela usando um arquivo JSON de exemplo.
- Consultar filmes que foram lançados em determinado ano.
- Verificar filmes que foram lançados em um intervalo de anos.
- Excluir um filme da tabela e, depois, excluir a tabela.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
/// <summary>
/// This example application performs the following basic Amazon DynamoDB
/// functions:
///     CreateTableAsync
///     PutItemAsync
///     UpdateItemAsync
///     BatchWriteItemAsync
///     GetItemAsync
///     DeleteItemAsync
///     Query
///     Scan
///     DeleteItemAsync.
/// </summary>
public class DynamoDbBasics
{
    public static bool IsInteractive = true;
```

```
    // Separator for the console display.
    private static readonly string SepBar = new string('-', 80);

    /// <summary>
    /// The main entry point for the DynamoDB Basics example application.
    /// </summary>
    /// <param name="args">Command line arguments.</param>
    /// <returns>A task representing the asynchronous operation.</returns>
    public static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon DynamoDB.
        using var host =
Microsoft.Extensions.Hosting.Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonDynamoDB>()
                    .AddTransient<DynamoDbWrapper>())
            .Build();

        // Now the wrapper is available for injection.
        var dynamoDbWrapper = host.Services.GetRequiredService<DynamoDbWrapper>();

        var tableName = "movie_table";

        var movieFileName = @"movies.json";

        DisplayInstructions();

        // Create a new table and wait for it to be active.
        Console.WriteLine($"Creating the new table: {tableName}");

        var success = await dynamoDbWrapper.CreateMovieTableAsync(tableName);

        Console.WriteLine(success
            ? $"\nTable: {tableName} successfully created."
            : $"\nCould not create {tableName}.");

        WaitForEnter();

        // Add a single new movie to the table.
        var newMovie = new Movie
        {
            Year = 2021,
            Title = "Spider-Man: No Way Home",
        };
```

```
        success = await dynamoDbWrapper.PutItemAsync(newMovie, tableName);
        if (success)
        {
            Console.WriteLine($"Added {newMovie.Title} to the table.");
        }
        else
        {
            Console.WriteLine("Could not add movie to table.");
        }

        WaitForEnter();

        // Update the new movie by adding a plot and rank.
        var newInfo = new MovieInfo
        {
            Plot = "With Spider-Man's identity now revealed, Peter asks" +
                   "Doctor Strange for help. When a spell goes wrong, dangerous" +
                   "foes from other worlds start to appear, forcing Peter to" +
                   "discover what it truly means to be Spider-Man.",
            Rank = 9,
        };

        success = await dynamoDbWrapper.UpdateItemAsync(newMovie, newInfo,
tableName);
        if (success)
        {
            Console.WriteLine($"Successfully updated the movie: {newMovie.Title}");
        }
        else
        {
            Console.WriteLine("Could not update the movie.");
        }

        WaitForEnter();

        // Add a batch of movies to the DynamoDB table from a list of
        // movies in a JSON file.
        var itemCount = await dynamoDbWrapper.BatchWriteItemsAsync(movieFileName,
tableName);
        Console.WriteLine($"Added {itemCount} movies to the table.");

        WaitForEnter();
```

```
        // Get a movie by key. (partition + sort)
        var lookupMovie = new Movie
        {
            Title = "Jurassic Park",
            Year = 1993,
        };

        Console.WriteLine("Looking for the movie \"Jurassic Park\".");
        var item = await dynamoDbWrapper.GetItemAsync(lookupMovie, tableName);
        if (item?.Count > 0)
        {
            dynamoDbWrapper.DisplayItem(item);
        }
        else
        {
            Console.WriteLine($"Couldn't find {lookupMovie.Title}");
        }

        WaitForEnter();

        // Delete a movie.
        var movieToDelete = new Movie
        {
            Title = "The Town",
            Year = 2010,
        };

        success = await dynamoDbWrapper.DeleteItemAsync(tableName, movieToDelete);

        if (success)
        {
            Console.WriteLine($"Successfully deleted {movieToDelete.Title}.");
        }
        else
        {
            Console.WriteLine($"Could not delete {movieToDelete.Title}.");
        }

        WaitForEnter();

        // Use Query to find all the movies released in 2010.
        int findYear = 2010;
        Console.WriteLine($"Movies released in {findYear}");
```

```
        var queryCount = await dynamoDbWrapper.QueryMoviesAsync(tableName,
findYear);
        Console.WriteLine($"Found {queryCount} movies released in {findYear}");

        WaitForEnter();

        // Use Scan to get a list of movies from 2001 to 2011.
        int startYear = 2001;
        int endYear = 2011;
        var scanCount = await dynamoDbWrapper.ScanTableAsync(tableName, startYear,
endYear);
        Console.WriteLine($"Found {scanCount} movies released between {startYear}
and {endYear}");

        WaitForEnter();

        // Delete the table.
        success = await dynamoDbWrapper.DeleteTableAsync(tableName);

        if (success)
        {
            Console.WriteLine($"Successfully deleted {tableName}");
        }
        else
        {
            Console.WriteLine($"Could not delete {tableName}");
        }

        Console.WriteLine("The DynamoDB Basics example application is complete.");

        WaitForEnter();
    }

    /// <summary>
    /// Displays the description of the application on the console.
    /// </summary>
    private static void DisplayInstructions()
    {
        if (!IsInteractive)
        {
            return;
        }

        Console.Clear();
```

```
        Console.WriteLine();
        Console.Write(new string(' ', 28));
        Console.WriteLine("DynamoDB Basics Example");
        Console.WriteLine(SepBar);
        Console.WriteLine("This demo application shows the basics of using DynamoDB
 with the AWS SDK.");
        Console.WriteLine(SepBar);
        Console.WriteLine("The application does the following:");
        Console.WriteLine("\t1. Creates a table with partition: year and
 sort:title.");
        Console.WriteLine("\t2. Adds a single movie to the table.");
        Console.WriteLine("\t3. Adds movies to the table from moviedata.json.");
        Console.WriteLine("\t4. Updates the rating and plot of the movie that was
 just added.");
        Console.WriteLine("\t5. Gets a movie using its key (partition + sort).");
        Console.WriteLine("\t6. Deletes a movie.");
        Console.WriteLine("\t7. Uses QueryAsync to return all movies released in a
 given year.");
        Console.WriteLine("\t8. Uses ScanAsync to return all movies released within
 a range of years.");
        Console.WriteLine("\t9. Finally, it deletes the table that was just
 created.");
        WaitForEnter();
    }

    /// <summary>
    /// Simple method to wait for the Enter key to be pressed.
    /// </summary>
    private static void WaitForEnter()
    {
        if (IsInteractive)
        {
            Console.WriteLine("\nPress <Enter> to continue.");
            Console.WriteLine(SepBar);
            _ = Console.ReadLine();
        }
    }
}
```

Usar o cliente injetado para operações de tabela.

```
using System.Text.Json;
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;
using Amazon.DynamoDBv2.DocumentModel;
using Amazon.DynamoDBv2.Model;

namespace DynamoDBActions;

/// <summary>
/// Methods of this class perform Amazon DynamoDB operations.
/// </summary>
public class DynamoDbWrapper
{
    private readonly IAmazonDynamoDB _amazonDynamoDB;

    /// <summary>
    /// Constructor for the DynamoDbWrapper class.
    /// </summary>
    /// <param name="amazonDynamoDB">The injected DynamoDB client.</param>
    public DynamoDbWrapper(IAmazonDynamoDB amazonDynamoDB)
    {
        _amazonDynamoDB = amazonDynamoDB;
    }
```

Cria uma tabela para conter dados de filmes.

```
    /// <summary>
    /// Creates a new Amazon DynamoDB table and then waits for the new
    /// table to become active.
    /// </summary>
    /// <param name="tableName">The name of the table to create.</param>
    /// <returns>A Boolean value indicating the success of the operation.</returns>
    public async Task<bool> CreateMovieTableAsync(string tableName)
    {
        try
        {
            var response = await _amazonDynamoDB.CreateTableAsync(new
 CreateTableRequest
            {
                TableName = tableName,
                AttributeDefinitions = new List<AttributeDefinition>()
```

```
            {
                new AttributeDefinition
                {
                    AttributeName = "title",
                    AttributeType = ScalarAttributeType.S,
                },
                new AttributeDefinition
                {
                    AttributeName = "year",
                    AttributeType = ScalarAttributeType.N,
                },
            },
            KeySchema = new List<KeySchemaElement>()
            {
                new KeySchemaElement
                {
                    AttributeName = "year",
                    KeyType = KeyType.HASH,
                },
                new KeySchemaElement
                {
                    AttributeName = "title",
                    KeyType = KeyType.RANGE,
                },
            },
            BillingMode = BillingMode.PAY_PER_REQUEST,
        });

        // Wait until the table is ACTIVE and then report success.
        Console.Write("Waiting for table to become active...");

        var request = new DescribeTableRequest
        {
            TableName = response.TableDescription.TableName,
        };

        TableStatus status;

        int sleepDuration = 2000;

        do
        {
            Thread.Sleep(sleepDuration);
```

```
                var describeTableResponse = await
 _amazonDynamoDB.DescribeTableAsync(request);
                status = describeTableResponse.Table.TableStatus;

                Console.Write(".");
            }
            while (status != "ACTIVE");

            return status == TableStatus.ACTIVE;
        }
        catch (ResourceInUseException ex)
        {
            Console.WriteLine($"Table {tableName} already exists. {ex.Message}");
            throw;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while creating
 table {tableName}. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while creating table {tableName}.
 {ex.Message}");
            throw;
        }
    }
```

Adiciona um único filme à tabela.

```

    /// <summary>
    /// Adds a new item to the table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing informtation for
    /// the movie to add to the table.</param>
    /// <param name="tableName">The name of the table where the item will be
 added.</param>
    /// <returns>A Boolean value that indicates the results of adding the item.</
returns>
```

```
    public async Task<bool> PutItemAsync(Movie newMovie, string tableName)
    {
        try
        {
            var item = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };

            var request = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
            };

            await _amazonDynamoDB.PutItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while putting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while putting item.
{ex.Message}");
            throw;
        }
    }
```

Atualiza um item único em uma tabela.

```
    /// <summary>
    /// Updates an existing item in the movies table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing information for
    /// the movie to update.</param>
    /// <param name="newInfo">A MovieInfo object that contains the
    /// information that will be changed.</param>
    /// <param name="tableName">The name of the table that contains the movie.</
param>
    /// <returns>A Boolean value that indicates the success of the operation.</
returns>
    public async Task<bool> UpdateItemAsync(
        Movie newMovie,
        MovieInfo newInfo,
        string tableName)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };
            var updates = new Dictionary<string, AttributeValueUpdate>
            {
                ["info.plot"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { S = newInfo.Plot },
                },

                ["info.rating"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { N = newInfo.Rank.ToString() },
                },
            };

            var request = new UpdateItemRequest
            {
                AttributeUpdates = updates,
                Key = key,
                TableName = tableName,
            };
```

```
            await _amazonDynamoDB.UpdateItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} or item was not found.
{ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while updating
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while updating item.
{ex.Message}");
            throw;
        }
    }
```

Recupera um único item de uma tabela de filmes.

```
    /// <summary>
    /// Gets information about an existing movie from the table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing information about
    /// the movie to retrieve.</param>
    /// <param name="tableName">The name of the table containing the movie.</param>
    /// <returns>A Dictionary object containing information about the item
    /// retrieved.</returns>
    public async Task<Dictionary<string, AttributeValue>> GetItemAsync(Movie
newMovie, string tableName)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
```

```
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };

            var request = new GetItemRequest
            {
                Key = key,
                TableName = tableName,
            };

            var response = await _amazonDynamoDB.GetItemAsync(request);
            return response.Item;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return new Dictionary<string, AttributeValue>();
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while getting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while getting item.
{ex.Message}");
            throw;
        }
    }
```

Grava um lote de itens na tabela de filmes.

```
    /// <summary>
    /// Loads the contents of a JSON file into a list of movies to be
    /// added to the DynamoDB table.
    /// </summary>
    /// <param name="movieFileName">The name of the JSON file.</param>
```

```
    /// <returns>A generic list of movie objects.</returns>
    public List<Movie> ImportMovies(string movieFileName)
    {
        var moviesList = new List<Movie>();
        if (!File.Exists(movieFileName))
        {
            return moviesList;
        }

        using var sr = new StreamReader(movieFileName);
        string json = sr.ReadToEnd();
        var allMovies = JsonSerializer.Deserialize<List<Movie>>(
            json,
            new JsonSerializerOptions
            {
                PropertyNameCaseInsensitive = true
            });

        // Now return the first 250 entries.
        if (allMovies != null && allMovies.Any())
        {
            moviesList = allMovies.GetRange(0, 250);
        }
        return moviesList;
    }

    /// <summary>
    /// Writes 250 items to the movie table.
    /// </summary>
    /// <param name="movieFileName">A string containing the full path to
    /// the JSON file containing movie data.</param>
    /// <param name="tableName">The name of the table to write items to.</param>
    /// <returns>A long integer value representing the number of movies
    /// imported from the JSON file.</returns>
    public async Task<long> BatchWriteItemsAsync(
        string movieFileName, string tableName)
    {
        try
        {
            var movies = ImportMovies(movieFileName);
            if (!movies.Any())
            {
                Console.WriteLine("Couldn't find the JSON file with movie data.");
                return 0;
```

```
            }

            var context = new DynamoDBContextBuilder()
                // Optional call to provide a specific instance of IAmazonDynamoDB
                .WithDynamoDBClient(() => _amazonDynamoDB)
                .Build();

            var movieBatch = context.CreateBatchWrite<Movie>(
                new BatchWriteConfig()
                {
                    OverrideTableName = tableName
                });
            movieBatch.AddPutItems(movies);

            Console.WriteLine("Adding imported movies to the table.");
            await movieBatch.ExecuteAsync();

            return movies.Count;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table was not found during batch write operation.
{ex.Message}");
            throw;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred during batch write
operation. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred during batch write operation.
{ex.Message}");
            throw;
        }
    }
```

Exclui um único item da tabela.

```
    /// <summary>
    /// Deletes a single item from a DynamoDB table.
    /// </summary>
    /// <param name="tableName">The name of the table from which the item
    /// will be deleted.</param>
    /// <param name="movieToDelete">A movie object containing the title and
    /// year of the movie to delete.</param>
    /// <returns>A Boolean value indicating the success or failure of the
    /// delete operation.</returns>
    public async Task<bool> DeleteItemAsync(
        string tableName,
        Movie movieToDelete)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = movieToDelete.Title },
                ["year"] = new AttributeValue { N = movieToDelete.Year.ToString() },
            };

            var request = new DeleteItemRequest { TableName = tableName, Key =
key, };

            await _amazonDynamoDB.DeleteItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while deleting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting item.
{ex.Message}");
```

```
            throw;
        }
    }
```

Consulta a tabela de filmes lançados em determinado ano.

```
/// <summary>
/// Queries the table for movies released in a particular year and
/// then displays the information for the movies returned.
/// </summary>
/// <param name="tableName">The name of the table to query.</param>
/// <param name="year">The release year for which we want to
/// view movies.</param>
/// <returns>The number of movies that match the query.</returns>
public async Task<int> QueryMoviesAsync(string tableName, int year)
{
    try
    {
        var movieTable = new TableBuilder(_amazonDynamoDB, tableName)
            .AddHashKey("year", DynamoDBEntryType.Numeric)
            .AddRangeKey("title", DynamoDBEntryType.String)
            .Build();

        var filter = new QueryFilter("year", QueryOperator.Equal, year);

        Console.WriteLine("\nFind movies released in: {year}:");

        var config = new QueryOperationConfig()
        {
            Limit = 10, // 10 items per page.
            Select = SelectValues.SpecificAttributes,
            AttributesToGet = new List<string>
            {
                "title",
                "year",
            },
            ConsistentRead = true,
            Filter = filter,
        };
```

```
            // Value used to track how many movies match the
            // supplied criteria.
            var moviesFound = 0;

            var search = movieTable.Query(config);
            do
            {
                var movieList = await search.GetNextSetAsync();
                moviesFound += movieList.Count;

                foreach (var movie in movieList)
                {
                    DisplayDocument(movie);
                }
            }
            while (!search.IsDone);

            return moviesFound;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return 0;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while querying
movies. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while querying movies.
{ex.Message}");
            throw;
        }
    }
```

Busca na tabela os filmes lançados em um intervalo de anos.

```
    /// <summary>
```

```
    /// Scans the table for movies released between the specified years.
    /// </summary>
    /// <param name="tableName">The name of the table to scan.</param>
    /// <param name="startYear">The starting year for the range.</param>
    /// <param name="endYear">The ending year for the range.</param>
    /// <returns>The number of movies found in the specified year range.</returns>
    public async Task<int> ScanTableAsync(
        string tableName,
        int startYear,
        int endYear)
    {
        try
        {
            var request = new ScanRequest
            {
                TableName = tableName,
                ExpressionAttributeNames = new Dictionary<string, string>
                {
                    { "#yr", "year" },
                },
                ExpressionAttributeValues = new Dictionary<string, AttributeValue>
                {
                    { ":y_a", new AttributeValue { N = startYear.ToString() } },
                    { ":y_z", new AttributeValue { N = endYear.ToString() } },
                },
                FilterExpression = "#yr between :y_a and :y_z",
                ProjectionExpression = "#yr, title, info.actors[0], info.directors,
info.running_time_secs",
                Limit = 10 // Set a limit to demonstrate using the LastEvaluatedKey.
            };

            // Keep track of how many movies were found.
            int foundCount = 0;

            var response = new ScanResponse();
            do
            {
                response = await _amazonDynamoDB.ScanAsync(request);
                foundCount += response.Items.Count;
                response.Items.ForEach(i => DisplayItem(i));
                request.ExclusiveStartKey = response.LastEvaluatedKey;
            }
            while (response?.LastEvaluatedKey?.Count > 0);
            return foundCount;
```

```
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return 0;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while scanning
table. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while scanning table.
{ex.Message}");
            throw;
        }
    }
```

Exclui a tabela de filmes.

```
    /// <summary>
    /// Deletes a DynamoDB table.
    /// </summary>
    /// <param name="tableName">The name of the table to delete.</param>
    /// <returns>A Boolean value indicating the success of the operation.</returns>
    public async Task<bool> DeleteTableAsync(string tableName)
    {
        try
        {
            var request = new DeleteTableRequest
            {
                TableName = tableName,
            };

            var response = await _amazonDynamoDB.DeleteTableAsync(request);

            Console.WriteLine($"Table {response.TableDescription.TableName}
successfully deleted.");
            return true;
```

```
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found and cannot be
deleted. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while deleting
table {tableName}. {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting table {tableName}.
{ex.Message}");
            return false;
        }
    }
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para .NET .
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Query
  - Scan
  - UpdateItem

Ações

`BatchWriteItem`

O código de exemplo a seguir mostra como usar `BatchWriteItem`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Grava um lote de itens na tabela de filmes.

```
/// <summary>
/// Loads the contents of a JSON file into a list of movies to be
/// added to the DynamoDB table.
/// </summary>
/// <param name="movieFileName">The name of the JSON file.</param>
/// <returns>A generic list of movie objects.</returns>
public List<Movie> ImportMovies(string movieFileName)
{
    var moviesList = new List<Movie>();
    if (!File.Exists(movieFileName))
    {
        return moviesList;
    }

    using var sr = new StreamReader(movieFileName);
    string json = sr.ReadToEnd();
    var allMovies = JsonSerializer.Deserialize<List<Movie>>(
        json,
        new JsonSerializerOptions
        {
            PropertyNameCaseInsensitive = true
        });

    // Now return the first 250 entries.
    if (allMovies != null && allMovies.Any())
    {
```

```
            moviesList = allMovies.GetRange(0, 250);
        }
        return moviesList;
    }

    /// <summary>
    /// Writes 250 items to the movie table.
    /// </summary>
    /// <param name="movieFileName">A string containing the full path to
    /// the JSON file containing movie data.</param>
    /// <param name="tableName">The name of the table to write items to.</param>
    /// <returns>A long integer value representing the number of movies
    /// imported from the JSON file.</returns>
    public async Task<long> BatchWriteItemsAsync(
        string movieFileName, string tableName)
    {
        try
        {
            var movies = ImportMovies(movieFileName);
            if (!movies.Any())
            {
                Console.WriteLine("Couldn't find the JSON file with movie data.");
                return 0;
            }

            var context = new DynamoDBContextBuilder()
                // Optional call to provide a specific instance of IAmazonDynamoDB
                .WithDynamoDBClient(() => _amazonDynamoDB)
                .Build();

            var movieBatch = context.CreateBatchWrite<Movie>(
                new BatchWriteConfig()
                {
                    OverrideTableName = tableName
                });
            movieBatch.AddPutItems(movies);

            Console.WriteLine("Adding imported movies to the table.");
            await movieBatch.ExecuteAsync();

            return movies.Count;
        }
        catch (ResourceNotFoundException ex)
        {
```

```
            Console.WriteLine($"Table was not found during batch write operation.
{ex.Message}");
            throw;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred during batch write
operation. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred during batch write operation.
{ex.Message}");
            throw;
        }
    }
```

- Para obter detalhes da API, consulte BatchWriteItema Referência AWS SDK para .NET da API.

`CreateTable`

O código de exemplo a seguir mostra como usar `CreateTable`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Creates a new Amazon DynamoDB table and then waits for the new
    /// table to become active.
    /// </summary>
    /// <param name="tableName">The name of the table to create.</param>
    /// <returns>A Boolean value indicating the success of the operation.</returns>
    public async Task<bool> CreateMovieTableAsync(string tableName)
```

```
    {
        try
        {
            var response = await _amazonDynamoDB.CreateTableAsync(new
CreateTableRequest
            {
                TableName = tableName,
                AttributeDefinitions = new List<AttributeDefinition>()
                {
                    new AttributeDefinition
                    {
                        AttributeName = "title",
                        AttributeType = ScalarAttributeType.S,
                    },
                    new AttributeDefinition
                    {
                        AttributeName = "year",
                        AttributeType = ScalarAttributeType.N,
                    },
                },
                KeySchema = new List<KeySchemaElement>()
                {
                    new KeySchemaElement
                    {
                        AttributeName = "year",
                        KeyType = KeyType.HASH,
                    },
                    new KeySchemaElement
                    {
                        AttributeName = "title",
                        KeyType = KeyType.RANGE,
                    },
                },
                BillingMode = BillingMode.PAY_PER_REQUEST,
            });

            // Wait until the table is ACTIVE and then report success.
            Console.Write("Waiting for table to become active...");

            var request = new DescribeTableRequest
            {
                TableName = response.TableDescription.TableName,
            };
```

```
            TableStatus status;

            int sleepDuration = 2000;

            do
            {
                Thread.Sleep(sleepDuration);

                var describeTableResponse = await
_amazonDynamoDB.DescribeTableAsync(request);
                status = describeTableResponse.Table.TableStatus;

                Console.Write(".");
            }
            while (status != "ACTIVE");

            return status == TableStatus.ACTIVE;
        }
        catch (ResourceInUseException ex)
        {
            Console.WriteLine($"Table {tableName} already exists. {ex.Message}");
            throw;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while creating
table {tableName}. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while creating table {tableName}.
{ex.Message}");
            throw;
        }
    }
```

- Para obter detalhes da API, consulte CreateTablea Referência AWS SDK para .NET da API.

`DeleteItem`

O código de exemplo a seguir mostra como usar `DeleteItem`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Deletes a single item from a DynamoDB table.
    /// </summary>
    /// <param name="tableName">The name of the table from which the item
    /// will be deleted.</param>
    /// <param name="movieToDelete">A movie object containing the title and
    /// year of the movie to delete.</param>
    /// <returns>A Boolean value indicating the success or failure of the
    /// delete operation.</returns>
    public async Task<bool> DeleteItemAsync(
        string tableName,
        Movie movieToDelete)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = movieToDelete.Title },
                ["year"] = new AttributeValue { N = movieToDelete.Year.ToString() },
            };

            var request = new DeleteItemRequest { TableName = tableName, Key =
key, };

            await _amazonDynamoDB.DeleteItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
```

```
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while deleting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting item.
{ex.Message}");
            throw;
        }
    }
```

- Para obter detalhes da API, consulte DeleteItema Referência AWS SDK para .NET da API.

`DeleteTable`

O código de exemplo a seguir mostra como usar `DeleteTable`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Deletes a DynamoDB table.
    /// </summary>
    /// <param name="tableName">The name of the table to delete.</param>
    /// <returns>A Boolean value indicating the success of the operation.</returns>
    public async Task<bool> DeleteTableAsync(string tableName)
    {
        try
        {
```

```
            var request = new DeleteTableRequest
            {
                TableName = tableName,
            };

            var response = await _amazonDynamoDB.DeleteTableAsync(request);

            Console.WriteLine($"Table {response.TableDescription.TableName}
successfully deleted.");
            return true;

        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found and cannot be
deleted. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while deleting
table {tableName}. {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while deleting table {tableName}.
{ex.Message}");
            return false;
        }
    }
```

- Para obter detalhes da API, consulte DeleteTablea Referência AWS SDK para .NET da API.

`GetItem`

O código de exemplo a seguir mostra como usar `GetItem`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
   /// <summary>
   /// Gets information about an existing movie from the table.
   /// </summary>
   /// <param name="newMovie">A Movie object containing information about
   /// the movie to retrieve.</param>
   /// <param name="tableName">The name of the table containing the movie.</param>
   /// <returns>A Dictionary object containing information about the item
   /// retrieved.</returns>
   public async Task<Dictionary<string, AttributeValue>> GetItemAsync(Movie
newMovie, string tableName)
   {
       try
       {
           var key = new Dictionary<string, AttributeValue>
           {
               ["title"] = new AttributeValue { S = newMovie.Title },
               ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
           };

           var request = new GetItemRequest
           {
               Key = key,
               TableName = tableName,
           };

           var response = await _amazonDynamoDB.GetItemAsync(request);
           return response.Item;
       }
       catch (ResourceNotFoundException ex)
       {
           Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
           return new Dictionary<string, AttributeValue>();
       }
```

```
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while getting
 item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while getting item.
 {ex.Message}");
            throw;
        }
    }
```

- Para obter detalhes da API, consulte GetItema Referência AWS SDK para .NET da API.

`PutItem`

O código de exemplo a seguir mostra como usar `PutItem`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Adds a new item to the table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing informtation for
    /// the movie to add to the table.</param>
    /// <param name="tableName">The name of the table where the item will be
 added.</param>
    /// <returns>A Boolean value that indicates the results of adding the item.</
returns>
    public async Task<bool> PutItemAsync(Movie newMovie, string tableName)
    {
```

```
        try
        {
            var item = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };

            var request = new PutItemRequest
            {
                TableName = tableName,
                Item = item,
            };

            await _amazonDynamoDB.PutItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while putting
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while putting item.
{ex.Message}");
            throw;
        }
    }
```

- Para obter detalhes da API, consulte PutItema Referência AWS SDK para .NET da API.

`Query`

O código de exemplo a seguir mostra como usar `Query`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
/// <summary>
/// Queries the table for movies released in a particular year and
/// then displays the information for the movies returned.
/// </summary>
/// <param name="tableName">The name of the table to query.</param>
/// <param name="year">The release year for which we want to
/// view movies.</param>
/// <returns>The number of movies that match the query.</returns>
public async Task<int> QueryMoviesAsync(string tableName, int year)
{
    try
    {
        var movieTable = new TableBuilder(_amazonDynamoDB, tableName)
            .AddHashKey("year", DynamoDBEntryType.Numeric)
            .AddRangeKey("title", DynamoDBEntryType.String)
            .Build();

        var filter = new QueryFilter("year", QueryOperator.Equal, year);

        Console.WriteLine("\nFind movies released in: {year}:");

        var config = new QueryOperationConfig()
        {
            Limit = 10, // 10 items per page.
            Select = SelectValues.SpecificAttributes,
            AttributesToGet = new List<string>
            {
                "title",
                "year",
            },
            ConsistentRead = true,
            Filter = filter,
        };
```

```
            // Value used to track how many movies match the
            // supplied criteria.
            var moviesFound = 0;

            var search = movieTable.Query(config);
            do
            {
                var movieList = await search.GetNextSetAsync();
                moviesFound += movieList.Count;

                foreach (var movie in movieList)
                {
                    DisplayDocument(movie);
                }
            }
            while (!search.IsDone);

            return moviesFound;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return 0;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while querying
movies. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while querying movies.
{ex.Message}");
            throw;
        }
    }
```

- Consulte detalhes da API em Query na Referência da API AWS SDK para .NET .

Scan

O código de exemplo a seguir mostra como usar `Scan`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Scans the table for movies released between the specified years.
    /// </summary>
    /// <param name="tableName">The name of the table to scan.</param>
    /// <param name="startYear">The starting year for the range.</param>
    /// <param name="endYear">The ending year for the range.</param>
    /// <returns>The number of movies found in the specified year range.</returns>
    public async Task<int> ScanTableAsync(
        string tableName,
        int startYear,
        int endYear)
    {
        try
        {
            var request = new ScanRequest
            {
                TableName = tableName,
                ExpressionAttributeNames = new Dictionary<string, string>
                {
                    { "#yr", "year" },
                },
                ExpressionAttributeValues = new Dictionary<string, AttributeValue>
                {
                    { ":y_a", new AttributeValue { N = startYear.ToString() } },
                    { ":y_z", new AttributeValue { N = endYear.ToString() } },
                },
                FilterExpression = "#yr between :y_a and :y_z",
                ProjectionExpression = "#yr, title, info.actors[0], info.directors,
info.running_time_secs",
                Limit = 10 // Set a limit to demonstrate using the LastEvaluatedKey.
            };
```

```
            // Keep track of how many movies were found.
            int foundCount = 0;

            var response = new ScanResponse();
            do
            {
                response = await _amazonDynamoDB.ScanAsync(request);
                foundCount += response.Items.Count;
                response.Items.ForEach(i => DisplayItem(i));
                request.ExclusiveStartKey = response.LastEvaluatedKey;
            }
            while (response?.LastEvaluatedKey?.Count > 0);
            return foundCount;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} was not found. {ex.Message}");
            return 0;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while scanning
table. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while scanning table.
{ex.Message}");
            throw;
        }
    }
```

- Consulte detalhes da API em Scan na Referência da API AWS SDK para .NET .

`UpdateItem`

O código de exemplo a seguir mostra como usar `UpdateItem`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Updates an existing item in the movies table.
    /// </summary>
    /// <param name="newMovie">A Movie object containing information for
    /// the movie to update.</param>
    /// <param name="newInfo">A MovieInfo object that contains the
    /// information that will be changed.</param>
    /// <param name="tableName">The name of the table that contains the movie.</
param>
    /// <returns>A Boolean value that indicates the success of the operation.</
returns>
    public async Task<bool> UpdateItemAsync(
        Movie newMovie,
        MovieInfo newInfo,
        string tableName)
    {
        try
        {
            var key = new Dictionary<string, AttributeValue>
            {
                ["title"] = new AttributeValue { S = newMovie.Title },
                ["year"] = new AttributeValue { N = newMovie.Year.ToString() },
            };
            var updates = new Dictionary<string, AttributeValueUpdate>
            {
                ["info.plot"] = new AttributeValueUpdate
                {
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { S = newInfo.Plot },
                },

                ["info.rating"] = new AttributeValueUpdate
                {
```

```
                    Action = AttributeAction.PUT,
                    Value = new AttributeValue { N = newInfo.Rank.ToString() },
                },
            };

            var request = new UpdateItemRequest
            {
                AttributeUpdates = updates,
                Key = key,
                TableName = tableName,
            };

            await _amazonDynamoDB.UpdateItemAsync(request);
            return true;
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Table {tableName} or item was not found.
{ex.Message}");
            return false;
        }
        catch (AmazonDynamoDBException ex)
        {
            Console.WriteLine($"An Amazon DynamoDB error occurred while updating
item. {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred while updating item.
{ex.Message}");
            throw;
        }
    }
```

- Para obter detalhes da API, consulte UpdateItema Referência AWS SDK para .NET da API.

Exemplos do Amazon EC2 usando SDK para .NET (v4)

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET (v4) com o Amazon EC2.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- Conceitos básicos

Conceitos básicos

Olá, Amazon EC2

O exemplo de código a seguir mostra como começar a usar o Amazon EC2.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
namespace EC2Actions;

public class HelloEc2
{
    /// <summary>
    /// HelloEc2 lists the existing security groups for the default users.
    /// </summary>
    /// <param name="args">Command line arguments</param>
    /// <returns>Async task.</returns>
    static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon Elastic Compute Cloud (Amazon
 EC2).
```

```
        using var host =
Microsoft.Extensions.Hosting.Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonEC2>()
                .AddTransient<EC2Wrapper>()
            )
            .Build();

        // Now the client is available for injection.
        var ec2Client = host.Services.GetRequiredService<IAmazonEC2>();

        try
        {
            // Retrieve information for up to 10 Amazon EC2 security groups.
            var request = new DescribeSecurityGroupsRequest { MaxResults = 10 };
            var securityGroups = new List<SecurityGroup>();

            var paginatorForSecurityGroups =
                ec2Client.Paginators.DescribeSecurityGroups(request);

            await foreach (var securityGroup in
paginatorForSecurityGroups.SecurityGroups)
            {
                securityGroups.Add(securityGroup);
            }

            // Now print the security groups returned by the call to
            // DescribeSecurityGroupsAsync.
            Console.WriteLine("Welcome to the EC2 Hello Service example. " +
                              "\nLet's list your Security Groups:");
            securityGroups.ForEach(group =>
            {
                Console.WriteLine(
                    $"Security group: {group.GroupName} ID: {group.GroupId}");
            });
        }
        catch (AmazonEC2Exception ex)
        {
            Console.WriteLine($"An Amazon EC2 service error occurred while listing
security groups. {ex.Message}");
        }
        catch (Exception ex)
        {
```

```
            Console.WriteLine($"An error occurred while listing security groups.
 {ex.Message}");
        }
    }
}
```

- Para obter detalhes da API, consulte DescribeSecurityGroupsa Referência AWS SDK para .NET da API.

Exemplos do Amazon ECS usando SDK para .NET (v4)

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET (v4) com o Amazon ECS.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- Conceitos básicos

Conceitos básicos

Olá, Amazon ECS

O exemplo de código a seguir mostra como começar a usar o Amazon ECS.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
using Amazon.ECS;
using Amazon.ECS.Model;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
```

```
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Logging.Console;
using Microsoft.Extensions.Logging.Debug;

namespace ECSActions;

/// <summary>
/// A class that introduces the Amazon ECS Client by listing the
/// cluster ARNs for the account.
/// </summary>
public class HelloECS
{
    static async System.Threading.Tasks.Task Main(string[] args)
    {
        // Use the AWS .NET Core Setup package to set up dependency injection for
 the Amazon ECS client.
        // Use your AWS profile name, or leave it blank to use the default profile.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
 LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft", LogLevel.Trace))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonECS>()
            )
            .Build();

        var amazonECSClient = host.Services.GetRequiredService<IAmazonECS>();

        Console.WriteLine($"Hello Amazon ECS! Following are some cluster ARNS
 available in the your account");
        Console.WriteLine();

        var clusters = new List<string>();

        var clustersPaginator = amazonECSClient.Paginators.ListClusters(new
 ListClustersRequest());

        await foreach (var response in clustersPaginator.Responses)
        {
            clusters.AddRange(response.ClusterArns);
        }
```

```
        if (clusters.Count > 0)
        {
            clusters.ForEach(cluster =>
            {
                Console.WriteLine($"\tARN: {cluster}");
                Console.WriteLine($"Cluster Name: {cluster.Split("/").Last()}");
                Console.WriteLine();
            });
        }
        else
        {
            Console.WriteLine("No clusters were found.");
        }

    }
}
```

- Para obter detalhes da API, consulte ListClusters a Referência AWS SDK para .NET da API.

AWS IoT exemplos usando SDK para .NET (v4)

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET (v4) com AWS IoT.

As noções básicas são exemplos de código que mostram como realizar as operações essenciais em um serviço.

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar perfis de serviço individuais, você pode ver as ações no contexto em seus cenários relacionados.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- Conceitos básicos
- Conceitos básicos
- Ações

Conceitos básicos

Olá AWS IoT

O exemplo de código a seguir mostra como começar a usar o AWS IoT.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
/// <summary>
/// Hello AWS IoT example.
/// </summary>
public class HelloIoT
{
    /// <summary>
    /// Main method to run the Hello IoT example.
    /// </summary>
    /// <param name="args">Command line arguments.</param>
    /// <returns>A Task object.</returns>
    public static async Task Main(string[] args)
    {
        var iotClient = new AmazonIoTClient();

        try
        {
            Console.WriteLine("Hello AWS IoT! Let's list your IoT Things:");
            Console.WriteLine(new string('-', 80));

            // Use pages of 10.
            var request = new ListThingsRequest()
            {
                MaxResults = 10
            };
            var response = await iotClient.ListThingsAsync(request);

            // Since there is not a built-in paginator, use the NextMarker to
 paginate.
```

```
            bool hasMoreResults = true;

            var things = new List<ThingAttribute>();
            while (hasMoreResults)
            {
                things.AddRange(response.Things);

                // If NextMarker is not null, there are more results. Get the next
page of results.
                if (!String.IsNullOrEmpty(response.NextMarker))
                {
                    request.Marker = response.NextMarker;
                    response = await iotClient.ListThingsAsync(request);
                }
                else
                    hasMoreResults = false;
            }

            if (things is { Count: > 0 })
            {
                Console.WriteLine($"Found {things.Count} IoT Things:");
                foreach (var thing in things)
                {
                    Console.WriteLine($"- Thing Name: {thing.ThingName}");
                    Console.WriteLine($"  Thing ARN: {thing.ThingArn}");
                    Console.WriteLine($"  Thing Type: {thing.ThingTypeName ?? "No
type specified"}");
                    Console.WriteLine($"  Version: {thing.Version}");

                    if (thing.Attributes?.Count > 0)
                    {
                        Console.WriteLine("  Attributes:");
                        foreach (var attr in thing.Attributes)
                        {
                            Console.WriteLine($"    {attr.Key}: {attr.Value}");
                        }
                    }
                    Console.WriteLine();
                }
            }
            else
            {
                Console.WriteLine("No IoT Things found in your account.");
```

```
                Console.WriteLine("You can create IoT Things using the IoT Basics
 scenario example.");
            }

            Console.WriteLine("Hello IoT completed successfully.");
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            Console.WriteLine($"Request throttled, please try again later:
 {ex.Message}");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't list Things. Here's why: {ex.Message}");
        }
    }
}
```

- Consulte detalhes da API em listThings na Referência da API do AWS SDK para .NET .

Conceitos básicos

Conheça os conceitos básicos

O exemplo de código a seguir mostra como:

- Crie qualquer AWS IoT coisa.
- Gerar um certificado de dispositivo.
- Atualize AWS IoT qualquer coisa com atributos.
- Exibir um endpoint exclusivo.
- Liste seus AWS IoT certificados.
- Atualize uma AWS IoT sombra.
- Gravar informações do estado.
- Cria uma regra.
- Listar suas regras.
- Pesquisar coisas usando o nome da coisa.
- Exclua qualquer AWS IoT coisa.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Execute um cenário interativo demonstrando AWS IoT recursos.

```
/// <summary>
/// Scenario class for AWS IoT basics.
/// </summary>
public class IoTBasics
{
    public static bool IsInteractive = true;
    public static IoTWrapper? Wrapper = null;
    public static IAmazonCloudFormation? CloudFormationClient = null;
    public static ILogger<IoTBasics> logger = null!;
    private static IoTWrapper _iotWrapper = null!;
    private static IAmazonCloudFormation _amazonCloudFormation = null!;
    private static ILogger<IoTBasics> _logger = null!;

    private static string _stackName = "IoTBasicsStack";
    private static string _stackResourcePath = "../../../../../../scenarios/basics/
iot/iot_usecase/resources/cfn_template.yaml";

    /// <summary>
    /// Main method for the IoT Basics scenario.
    /// </summary>
    /// <param name="args">Command line arguments.</param>
    /// <returns>A Task object.</returns>
    public static async Task Main(string[] args)
    {
        // Set up dependency injection for the Amazon service.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonIoT>(new AWSOptions() { Region =
 RegionEndpoint.USEast1 })
                    .AddAWSService<IAmazonCloudFormation>()
                        .AddTransient<IoTWrapper>()
                        .AddLogging(builder => builder.AddConsole())
                        .AddSingleton<IAmazonIotData>(sp =>
```

```
                        {
                            var iotService = sp.GetRequiredService<IAmazonIoT>();
                            var request = new DescribeEndpointRequest
                            {
                                EndpointType = "iot:Data-ATS"
                            };
                            var response =
 iotService.DescribeEndpointAsync(request).Result;
                            return new AmazonIotDataClient($"https://
{response.EndpointAddress}/");
                        })
            )
            .Build();

        logger = LoggerFactory.Create(builder => builder.AddConsole())
            .CreateLogger<IoTBasics>();

        Wrapper = host.Services.GetRequiredService<IoTWrapper>();
        CloudFormationClient =
 host.Services.GetRequiredService<IAmazonCloudFormation>();

        // Set the private fields for backwards compatibility
        _logger = logger;
        _iotWrapper = Wrapper;
        _amazonCloudFormation = CloudFormationClient;

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Welcome to the AWS IoT example scenario.");
        Console.WriteLine("This example program demonstrates various interactions
 with the AWS Internet of Things (IoT) Core service.");
        Console.WriteLine();
        if (IsInteractive)
        {
            Console.WriteLine("Press Enter to continue...");
            Console.ReadLine();
        }
        Console.WriteLine(new string('-', 80));

        try
        {
            await RunScenarioAsync();
        }
        catch (Exception ex)
        {
```

```
            _logger.LogError(ex, "There was a problem running the scenario.");
            Console.WriteLine($"\nAn error occurred: {ex.Message}");
        }

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("The AWS IoT scenario has successfully completed.");
        Console.WriteLine(new string('-', 80));
    }

    /// <summary>
    /// Run the IoT Basics scenario.
    /// </summary>
    /// <returns>A Task object.</returns>
    public static async Task RunScenarioAsync()
    {
        // Use static properties if available, otherwise use private fields
        var iotWrapper = Wrapper ?? _iotWrapper;
        var cloudFormationClient = CloudFormationClient ?? _amazonCloudFormation;
        var scenarioLogger = logger ?? _logger;

        await RunScenarioInternalAsync(iotWrapper, cloudFormationClient,
 scenarioLogger);
    }

    /// <summary>
    /// Internal method to run the IoT Basics scenario with injected dependencies.
    /// </summary>
    /// <param name="iotWrapper">The IoT wrapper instance.</param>
    /// <param name="cloudFormationClient">The CloudFormation client instance.</
param>
    /// <param name="scenarioLogger">The logger instance.</param>
    /// <returns>A Task object.</returns>
    private static async Task RunScenarioInternalAsync(IoTWrapper iotWrapper,
 IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics> scenarioLogger)
    {
        string thingName = $"iot-thing-{Guid.NewGuid():N}";
        string certificateArn = "";
        string certificateId = "";
        string ruleName = $"iotruledefault";
        string snsTopicArn = "";

        try
        {
            // Step 1: Create an AWS IoT Thing
```

```
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("1. Create an AWS IoT Thing.");
            Console.WriteLine("An AWS IoT Thing represents a virtual entity in the
AWS IoT service that can be associated with a physical device.");
            Console.WriteLine();

            if (IsInteractive)
            {
                Console.Write("Enter Thing name: ");
                var userInput = Console.ReadLine();
                if (!string.IsNullOrEmpty(userInput))
                    thingName = userInput;
            }
            else
            {
                Console.WriteLine($"Using default Thing name: {thingName}");
            }

            var thingArn = await iotWrapper.CreateThingAsync(thingName);
            Console.WriteLine($"{thingName} was successfully created. The ARN value
is {thingArn}");
            Console.WriteLine(new string('-', 80));

            // Step 1.1: List AWS IoT Things
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("2. List AWS IoT Things.");
            Console.WriteLine("Now let's list the IoT Things to see the Thing we
just created.");
            Console.WriteLine();
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var things = await iotWrapper.ListThingsAsync();
            Console.WriteLine($"Found {things.Count} IoT Things:");
            foreach (var thing in things.Take(10)) // Show first 10 things
            {
                Console.WriteLine($"Thing Name: {thing.ThingName}");
                Console.WriteLine($"Thing ARN: {thing.ThingArn}");
                if (thing.Attributes != null && thing.Attributes.Any())
                {
                    Console.WriteLine("Attributes:");
```

```
                    foreach (var attr in thing.Attributes)
                    {
                        Console.WriteLine($"  {attr.Key}: {attr.Value}");
                    }
                }
                Console.WriteLine("--------------");
            }
            Console.WriteLine();
            Console.WriteLine(new string('-', 80));

            // Step 2: Generate a Device Certificate
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("3. Generate a device certificate.");
            Console.WriteLine("A device certificate performs a role in securing the
communication between devices (Things) and the AWS IoT platform.");
            Console.WriteLine();

            var createCert = "y";
            if (IsInteractive)
            {
                Console.Write($"Do you want to create a certificate for {thingName}?
(y/n)");
                createCert = Console.ReadLine();
            }
            else
            {
                Console.WriteLine($"Creating certificate for {thingName}...");
            }

            if (createCert?.ToLower() == "y")
            {
                var certificateResult = await
iotWrapper.CreateKeysAndCertificateAsync();
                if (certificateResult.HasValue)
                {
                    var (certArn, certPem, certId) = certificateResult.Value;
                    certificateArn = certArn;
                    certificateId = certId;

                    Console.WriteLine($"\nCertificate:");
                    // Show only first few lines of certificate for brevity
                    var lines = certPem.Split('\n');
                    for (int i = 0; i < Math.Min(lines.Length, 5); i++)
                    {
```

```
                    Console.WriteLine(lines[i]);
                }
                if (lines.Length > 5)
                {
                    Console.WriteLine("...");
                }

                Console.WriteLine($"\nCertificate ARN:");
                Console.WriteLine(certificateArn);

                // Step 3: Attach the Certificate to the AWS IoT Thing
                Console.WriteLine("Attach the certificate to the AWS IoT
Thing.");
                var attachResult = await
iotWrapper.AttachThingPrincipalAsync(thingName, certificateArn);
                if (attachResult)
                {
                    Console.WriteLine("Certificate attached to Thing
successfully.");
                }
                else
                {
                    Console.WriteLine("Failed to attach certificate to Thing.");
                }

                Console.WriteLine("Thing Details:");
                Console.WriteLine($"Thing Name: {thingName}");
                Console.WriteLine($"Thing ARN: {thingArn}");
            }
            else
            {
                Console.WriteLine("Failed to create certificate.");
            }
        }
        Console.WriteLine(new string('-', 80));

        // Step 4: Update an AWS IoT Thing with Attributes
        Console.WriteLine(new string('-', 80));
        Console.WriteLine("4. Update an AWS IoT Thing with Attributes.");
        Console.WriteLine("IoT Thing attributes, represented as key-value pairs,
offer a pivotal advantage in facilitating efficient data");
        Console.WriteLine("management and retrieval within the AWS IoT
ecosystem.");
        Console.WriteLine();
```

```
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var attributes = new Dictionary<string, string>
            {
                { "Location", "Seattle" },
                { "DeviceType", "Sensor" },
                { "Firmware", "1.2.3" }
            };

            await iotWrapper.UpdateThingAsync(thingName, attributes);
            Console.WriteLine("Thing attributes updated successfully.");
            Console.WriteLine(new string('-', 80));

            // Step 5: Return a unique endpoint specific to the Amazon Web Services
account
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("5. Return a unique endpoint specific to the Amazon
Web Services account.");
            Console.WriteLine();
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var endpoint = await iotWrapper.DescribeEndpointAsync();
            if (endpoint != null)
            {
                var subdomain = endpoint.Split('.')[0];
                Console.WriteLine($"Extracted subdomain: {subdomain}");
                Console.WriteLine($"Full Endpoint URL: https://{endpoint}");
            }
            else
            {
                Console.WriteLine("Failed to retrieve endpoint.");
            }
            Console.WriteLine(new string('-', 80));

            // Step 6: List your AWS IoT certificates
            Console.WriteLine(new string('-', 80));
```

```
            Console.WriteLine("6. List your AWS IoT certificates");
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var certificates = await iotWrapper.ListCertificatesAsync();
            foreach (var cert in certificates.Take(5)) // Show first 5 certificates
            {
                Console.WriteLine($"Cert id: {cert.CertificateId}");
                Console.WriteLine($"Cert Arn: {cert.CertificateArn}");
            }
            Console.WriteLine();
            Console.WriteLine(new string('-', 80));

            // Step 7: Create an IoT shadow
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("7. Update an IoT shadow that refers to a digital
representation or virtual twin of a physical IoT device");
            Console.WriteLine();
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var shadowPayload = JsonSerializer.Serialize(new
            {
                state = new
                {
                    desired = new
                    {
                        temperature = 25,
                        humidity = 50
                    }
                }
            });

            await iotWrapper.UpdateThingShadowAsync(thingName, shadowPayload);
            Console.WriteLine("Thing Shadow updated successfully.");
            Console.WriteLine(new string('-', 80));

            // Step 8: Write out the state information, in JSON format
```

```
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("8. Write out the state information, in JSON
format.");
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var shadowData = await iotWrapper.GetThingShadowAsync(thingName);
            Console.WriteLine($"Received Shadow Data: {shadowData}");
            Console.WriteLine(new string('-', 80));

            // Step 9: Set up resources (SNS topic and IAM role) and create a rule
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("9. Set up resources and create a rule");
            Console.WriteLine();

            // Deploy CloudFormation stack to create SNS topic and IAM role
            Console.WriteLine("Deploying CloudFormation stack to create SNS topic
and IAM role...");

            var deployStack = !IsInteractive || GetYesNoResponse("Would you like to
deploy the CloudFormation stack? (y/n) ");
            if (deployStack)
            {
                if (IsInteractive)
                {
                    Console.Write(
                        $"Enter stack resource file path (or press Enter for default
'{_stackResourcePath}'): ");
                    var userResourcePath = Console.ReadLine();
                    if (!string.IsNullOrEmpty(userResourcePath))
                        _stackResourcePath = userResourcePath;
                }

                _stackName = PromptUserForStackName();

                var deploySuccess = await DeployCloudFormationStack(_stackName,
cloudFormationClient, scenarioLogger);

                if (deploySuccess)
                {
                    // Get stack outputs
```

```
                    var stackOutputs = await GetStackOutputs(_stackName,
cloudFormationClient, scenarioLogger);
                    if (stackOutputs != null)
                    {
                        snsTopicArn = stackOutputs["SNSTopicArn"];
                        string roleArn = stackOutputs["RoleArn"];

                        Console.WriteLine($"Successfully deployed stack. SNS topic:
{snsTopicArn}");
                        Console.WriteLine($"Successfully deployed stack. IAM role:
{roleArn}");

                        if (IsInteractive)
                        {
                            Console.Write($"Enter Rule name (press Enter for default
'{ruleName}'): ");
                            var userRuleName = Console.ReadLine();
                            if (!string.IsNullOrEmpty(userRuleName))
                                ruleName = userRuleName;
                        }
                        else
                        {
                            Console.WriteLine($"Using default rule name:
{ruleName}");
                        }

                        // Now create the IoT rule with the CloudFormation outputs
                        var ruleResult = await
iotWrapper.CreateTopicRuleAsync(ruleName, snsTopicArn, roleArn);
                        if (ruleResult)
                        {
                            Console.WriteLine("IoT Rule created successfully.");
                        }
                        else
                        {
                            Console.WriteLine("Failed to create IoT rule.");
                        }
                    }
                    else
                    {
                        Console.WriteLine("Failed to get stack outputs. Skipping
rule creation.");
                    }
                }
```

```
                else
                {
                    Console.WriteLine("Failed to deploy CloudFormation stack.
 Skipping rule creation.");
                }
            }
            else
            {
                Console.WriteLine("Skipping CloudFormation stack deployment and rule
 creation.");
            }
            Console.WriteLine(new string('-', 80));

            // Step 10: List your rules
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("10. List your rules.");
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var rules = await iotWrapper.ListTopicRulesAsync();
            Console.WriteLine("List of IoT Rules:");
            foreach (var rule in rules.Take(5)) // Show first 5 rules
            {
                Console.WriteLine($"Rule Name: {rule.RuleName}");
                Console.WriteLine($"Rule ARN: {rule.RuleArn}");
                Console.WriteLine("--------------");
            }
            Console.WriteLine();
            Console.WriteLine(new string('-', 80));

            // Step 11: Search things using the Thing name
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("11. Search things using the Thing name.");
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

            var searchResults = await iotWrapper.SearchIndexAsync($"thingName:
{thingName}");
```

```
            if (searchResults.Any())
            {
                Console.WriteLine($"Thing id found using search is
{searchResults.First().ThingId}");
            }
            else
            {
                Console.WriteLine($"No search results found for Thing:
{thingName}");
            }
            Console.WriteLine(new string('-', 80));

            // Step 12: Cleanup - Detach and delete certificate
            if (!string.IsNullOrEmpty(certificateArn))
            {
                Console.WriteLine(new string('-', 80));
                var deleteCert = "y";
                if (IsInteractive)
                {
                    Console.Write($"Do you want to detach and delete the certificate
for {thingName}? (y/n)");
                    deleteCert = Console.ReadLine();
                }
                else
                {
                    Console.WriteLine($"Detaching and deleting certificate for
{thingName}...");
                }

                if (deleteCert?.ToLower() == "y")
                {
                    Console.WriteLine("12. You selected to detach and delete the
certificate.");
                    if (IsInteractive)
                    {
                        Console.WriteLine("Press Enter to continue...");
                        Console.ReadLine();
                    }

                    await iotWrapper.DetachThingPrincipalAsync(thingName,
certificateArn);
                    Console.WriteLine($"{certificateArn} was successfully removed
from {thingName}");
```

```
                    await iotWrapper.DeleteCertificateAsync(certificateId);
                    Console.WriteLine($"{certificateArn} was successfully
deleted.");
                }
                Console.WriteLine(new string('-', 80));
            }

            // Step 13: Delete the AWS IoT Thing
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("13. Delete the AWS IoT Thing.");
            var deleteThing = "y";
            if (IsInteractive)
            {
                Console.Write($"Do you want to delete the IoT Thing? (y/n)");
                deleteThing = Console.ReadLine();
            }
            else
            {
                Console.WriteLine($"Deleting IoT Thing {thingName}...");
            }

            if (deleteThing?.ToLower() == "y")
            {
                await iotWrapper.DeleteThingAsync(thingName);
                Console.WriteLine($"Deleted Thing {thingName}");
            }
            Console.WriteLine(new string('-', 80));

            // Step 14: Clean up CloudFormation stack
            if (!string.IsNullOrEmpty(snsTopicArn))
            {
                Console.WriteLine(new string('-', 80));
                Console.WriteLine("14. Clean up CloudFormation stack.");
                Console.WriteLine("Deleting the CloudFormation stack and all
resources...");

                var cleanup = !IsInteractive || GetYesNoResponse("Do you want to
delete the CloudFormation stack and all resources? (y/n) ");
                if (cleanup)
                {
                    var ruleCleanupSuccess = await
iotWrapper.DeleteTopicRuleAsync(ruleName);
```

```
                    var stackCleanupSuccess = await
 DeleteCloudFormationStack(_stackName, cloudFormationClient, scenarioLogger);
                    if (ruleCleanupSuccess && stackCleanupSuccess)
                    {
                        Console.WriteLine("Successfully cleaned up CloudFormation
 stack and all resources.");
                    }
                    else
                    {
                        Console.WriteLine("Some cleanup operations failed. Check the
 logs for details.");
                    }
                }
                else
                {
                    Console.WriteLine($"Resources will remain. Stack name:
 {_stackName}");
                }
                Console.WriteLine(new string('-', 80));
            }
        }
        catch (Exception ex)
        {
            scenarioLogger.LogError(ex, "Error occurred during scenario
 execution.");

            // Cleanup on error
            if (!string.IsNullOrEmpty(certificateArn) && !
string.IsNullOrEmpty(thingName))
            {
                try
                {
                    await iotWrapper.DetachThingPrincipalAsync(thingName,
 certificateArn);
                    await iotWrapper.DeleteCertificateAsync(certificateId);
                }
                catch (Exception cleanupEx)
                {
                    scenarioLogger.LogError(cleanupEx, "Error during cleanup.");
                }
            }

            if (!string.IsNullOrEmpty(thingName))
            {
```

```
                try
                {
                    await iotWrapper.DeleteThingAsync(thingName);
                }
                catch (Exception cleanupEx)
                {
                    scenarioLogger.LogError(cleanupEx, "Error during Thing
cleanup.");
                }
            }

            // Clean up CloudFormation stack on error
            if (!string.IsNullOrEmpty(snsTopicArn))
            {
                try
                {
                    await _iotWrapper.DeleteTopicRuleAsync(ruleName);
                    await DeleteCloudFormationStack(_stackName,
cloudFormationClient, scenarioLogger);
                }
                catch (Exception cleanupEx)
                {
                    scenarioLogger.LogError(cleanupEx, "Error during CloudFormation
stack cleanup.");
                }
            }

            throw;
        }
    }

    /// <summary>
    /// Deploys the CloudFormation stack with the necessary resources.
    /// </summary>
    /// <param name="stackName">The name of the CloudFormation stack.</param>
    /// <param name="cloudFormationClient">The CloudFormation client.</param>
    /// <param name="scenarioLogger">The logger.</param>
    /// <returns>True if the stack was deployed successfully.</returns>
    private static async Task<bool> DeployCloudFormationStack(string stackName,
IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics> scenarioLogger)
    {
        Console.WriteLine($"\nDeploying CloudFormation stack: {stackName}");

        try
```

```
        {
            var request = new CreateStackRequest
            {
                StackName = stackName,
                TemplateBody = await File.ReadAllTextAsync(_stackResourcePath),
                Capabilities = new List<string> { Capability.CAPABILITY_NAMED_IAM }
            };

            var response = await cloudFormationClient.CreateStackAsync(request);

            if (response.HttpStatusCode == System.Net.HttpStatusCode.OK)
            {
                Console.WriteLine($"CloudFormation stack creation started:
{stackName}");

                bool stackCreated = await WaitForStackCompletion(response.StackId,
cloudFormationClient, scenarioLogger);

                if (stackCreated)
                {
                    Console.WriteLine("CloudFormation stack created successfully.");
                    return true;
                }
                else
                {
                    scenarioLogger.LogError($"CloudFormation stack creation failed:
{stackName}");
                    return false;
                }
            }
            else
            {
                scenarioLogger.LogError($"Failed to create CloudFormation stack:
{stackName}");
                return false;
            }
        }
        catch (AlreadyExistsException)
        {
            scenarioLogger.LogWarning($"CloudFormation stack '{stackName}' already
exists. Please provide a unique name.");
            var newStackName = PromptUserForStackName();
            return await DeployCloudFormationStack(newStackName,
cloudFormationClient, scenarioLogger);
```

```
        }
        catch (Exception ex)
        {
            scenarioLogger.LogError(ex, $"An error occurred while deploying the
CloudFormation stack: {stackName}");
            return false;
        }
    }

    /// <summary>
    /// Waits for the CloudFormation stack to be in the CREATE_COMPLETE state.
    /// </summary>
    /// <param name="stackId">The ID of the CloudFormation stack.</param>
    /// <param name="cloudFormationClient">The CloudFormation client.</param>
    /// <param name="scenarioLogger">The logger.</param>
    /// <returns>True if the stack was created successfully.</returns>
    private static async Task<bool> WaitForStackCompletion(string stackId,
IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics> scenarioLogger)
    {
        int retryCount = 0;
        const int maxRetries = 30;
        const int retryDelay = 10000;

        while (retryCount < maxRetries)
        {
            var describeStacksRequest = new DescribeStacksRequest
            {
                StackName = stackId
            };

            var describeStacksResponse = await
cloudFormationClient.DescribeStacksAsync(describeStacksRequest);

            if (describeStacksResponse.Stacks.Count > 0)
            {
                if (describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.CREATE_COMPLETE)
                {
                    return true;
                }
                if (describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.CREATE_FAILED ||
                    describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.ROLLBACK_COMPLETE)
```

```
                {
                    return false;
                }
            }

            Console.WriteLine("Waiting for CloudFormation stack creation to
complete...");
            await Task.Delay(retryDelay);
            retryCount++;
        }

        scenarioLogger.LogError("Timed out waiting for CloudFormation stack creation
to complete.");
        return false;
    }

    /// <summary>
    /// Gets the outputs from the CloudFormation stack.
    /// </summary>
    /// <param name="stackName">The name of the CloudFormation stack.</param>
    /// <param name="cloudFormationClient">The CloudFormation client.</param>
    /// <param name="scenarioLogger">The logger.</param>
    /// <returns>A dictionary of stack outputs.</returns>
    private static async Task<Dictionary<string, string>?> GetStackOutputs(string
stackName, IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics>
scenarioLogger)
    {
        try
        {
            var describeStacksRequest = new DescribeStacksRequest
            {
                StackName = stackName
            };

            var response = await
cloudFormationClient.DescribeStacksAsync(describeStacksRequest);

            if (response.Stacks.Count > 0)
            {
                var outputs = new Dictionary<string, string>();
                foreach (var output in response.Stacks[0].Outputs)
                {
                    outputs[output.OutputKey] = output.OutputValue;
                }
```

```
                return outputs;
            }

            return null;
        }
        catch (Exception ex)
        {
            scenarioLogger.LogError(ex, $"Failed to get stack outputs for
{stackName}");
            return null;
        }
    }

    /// <summary>
    /// Deletes the CloudFormation stack and waits for confirmation.
    /// </summary>
    /// <param name="stackName">The name of the CloudFormation stack.</param>
    /// <param name="cloudFormationClient">The CloudFormation client.</param>
    /// <param name="scenarioLogger">The logger.</param>
    /// <returns>True if the stack was deleted successfully.</returns>
    private static async Task<bool> DeleteCloudFormationStack(string stackName,
IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics> scenarioLogger)
    {
        try
        {
            var request = new DeleteStackRequest
            {
                StackName = stackName
            };

            await cloudFormationClient.DeleteStackAsync(request);
            Console.WriteLine($"CloudFormation stack '{stackName}' is being deleted.
This may take a few minutes.");

            bool stackDeleted = await WaitForStackDeletion(stackName,
cloudFormationClient, scenarioLogger);

            if (stackDeleted)
            {
                Console.WriteLine($"CloudFormation stack '{stackName}' has been
deleted.");
                return true;
            }
            else
```

```
            {
                scenarioLogger.LogError($"Failed to delete CloudFormation stack
'{stackName}'.");
                return false;
            }
        }
        catch (Exception ex)
        {
            scenarioLogger.LogError(ex, $"An error occurred while deleting the
CloudFormation stack: {stackName}");
            return false;
        }
    }

    /// <summary>
    /// Waits for the stack to be deleted.
    /// </summary>
    /// <param name="stackName">The name of the CloudFormation stack.</param>
    /// <param name="cloudFormationClient">The CloudFormation client.</param>
    /// <param name="scenarioLogger">The logger.</param>
    /// <returns>True if the stack was deleted successfully.</returns>
    private static async Task<bool> WaitForStackDeletion(string stackName,
IAmazonCloudFormation cloudFormationClient, ILogger<IoTBasics> scenarioLogger)
    {
        int retryCount = 0;
        const int maxRetries = 30;
        const int retryDelay = 10000;

        while (retryCount < maxRetries)
        {
            var describeStacksRequest = new DescribeStacksRequest
            {
                StackName = stackName
            };

            try
            {
                var describeStacksResponse = await
cloudFormationClient.DescribeStacksAsync(describeStacksRequest);

                if (describeStacksResponse.Stacks.Count == 0 ||
                    describeStacksResponse.Stacks[0].StackStatus ==
StackStatus.DELETE_COMPLETE)
                {
```

```
                    return true;
                }
            }
            catch (AmazonCloudFormationException ex) when (ex.ErrorCode ==
"ValidationError")
            {
                return true;
            }

            Console.WriteLine($"Waiting for CloudFormation stack '{stackName}' to be
deleted...");
            await Task.Delay(retryDelay);
            retryCount++;
        }

        scenarioLogger.LogError($"Timed out waiting for CloudFormation stack
'{stackName}' to be deleted.");
        return false;
    }

    /// <summary>
    /// Helper method to get a yes or no response from the user.
    /// </summary>
    private static bool GetYesNoResponse(string question)
    {
        Console.WriteLine(question);
        var ynResponse = Console.ReadLine();
        var response = ynResponse != null && ynResponse.Equals("y",
StringComparison.InvariantCultureIgnoreCase);
        return response;
    }

    /// <summary>
    /// Prompts the user for a stack name.
    /// </summary>
    private static string PromptUserForStackName()
    {
        if (IsInteractive)
        {
            Console.Write($"Enter a name for the CloudFormation stack (press Enter
for default '{_stackName}'): ");
            string? input = Console.ReadLine();
            if (!string.IsNullOrWhiteSpace(input))
            {
```

```
                var regex = new System.Text.RegularExpressions.Regex("[a-zA-Z][-a-
zA-Z0-9]*");
                if (!regex.IsMatch(input))
                {
                    Console.WriteLine($"Invalid stack name. Using default:
 {_stackName}");
                    return _stackName;
                }
                return input;
            }
        }
        return _stackName;
    }
}
```

Uma classe de invólucro para métodos do AWS IoT SDK.

```
/// <summary>
/// Wrapper methods to use Amazon IoT Core with .NET.
/// </summary>
public class IoTWrapper
{
    private readonly IAmazonIoT _amazonIoT;
    private readonly IAmazonIotData _amazonIotData;
    private readonly ILogger<IoTWrapper> _logger;

    /// <summary>
    /// Constructor for the IoT wrapper.
    /// </summary>
    /// <param name="amazonIoT">The injected IoT client.</param>
    /// <param name="amazonIotData">The injected IoT Data client.</param>
    /// <param name="logger">The injected logger.</param>
    public IoTWrapper(IAmazonIoT amazonIoT, IAmazonIotData amazonIotData,
 ILogger<IoTWrapper> logger)
    {
        _amazonIoT = amazonIoT;
        _amazonIotData = amazonIotData;
        _logger = logger;
    }

    /// <summary>
    /// Creates an AWS IoT Thing.
```

```
    /// </summary>
    /// <param name="thingName">The name of the Thing to create.</param>
    /// <returns>The ARN of the Thing created, or null if creation failed.</returns>
    public async Task<string?> CreateThingAsync(string thingName)
    {
        try
        {
            var request = new CreateThingRequest
            {
                ThingName = thingName
            };

            var response = await _amazonIoT.CreateThingAsync(request);
            _logger.LogInformation($"Created Thing {thingName} with ARN
{response.ThingArn}");
            return response.ThingArn;
        }
        catch (Amazon.IoT.Model.ResourceAlreadyExistsException ex)
        {
            _logger.LogWarning($"Thing {thingName} already exists: {ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't create Thing {thingName}. Here's why:
{ex.Message}");
            return null;
        }
    }

    /// <summary>
    /// Creates a device certificate for AWS IoT.
    /// </summary>
    /// <returns>The certificate details including ARN and certificate PEM, or null
if creation failed.</returns>
    public async Task<(string CertificateArn, string CertificatePem, string
CertificateId)?> CreateKeysAndCertificateAsync()
    {
        try
        {
            var request = new CreateKeysAndCertificateRequest
            {
                SetAsActive = true
            };
```

```
            var response = await _amazonIoT.CreateKeysAndCertificateAsync(request);
            _logger.LogInformation($"Created certificate with ARN
{response.CertificateArn}");
            return (response.CertificateArn, response.CertificatePem,
response.CertificateId);
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't create certificate. Here's why:
{ex.Message}");
            return null;
        }
    }

    /// <summary>
    /// Attaches a certificate to an IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
    /// <param name="certificateArn">The ARN of the certificate to attach.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> AttachThingPrincipalAsync(string thingName, string
certificateArn)
    {
        try
        {
            var request = new AttachThingPrincipalRequest
            {
                ThingName = thingName,
                Principal = certificateArn
            };

            await _amazonIoT.AttachThingPrincipalAsync(request);
            _logger.LogInformation($"Attached certificate {certificateArn} to Thing
{thingName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
```

```
        {
            _logger.LogError($"Cannot attach certificate - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't attach certificate to Thing. Here's why:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Updates an IoT Thing with attributes.
    /// </summary>
    /// <param name="thingName">The name of the Thing to update.</param>
    /// <param name="attributes">Dictionary of attributes to add.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> UpdateThingAsync(string thingName, Dictionary<string,
string> attributes)
    {
        try
        {
            var request = new UpdateThingRequest
            {
                ThingName = thingName,
                AttributePayload = new AttributePayload
                {
                    Attributes = attributes,
                    Merge = true
                }
            };

            await _amazonIoT.UpdateThingAsync(request);
            _logger.LogInformation($"Updated Thing {thingName} with attributes");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot update Thing - resource not found:
{ex.Message}");
            return false;
        }
```

```
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't update Thing attributes. Here's why:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Gets the AWS IoT endpoint URL.
    /// </summary>
    /// <returns>The endpoint URL, or null if retrieval failed.</returns>
    public async Task<string?> DescribeEndpointAsync()
    {
        try
        {
            var request = new DescribeEndpointRequest
            {
                EndpointType = "iot:Data-ATS"
            };

            var response = await _amazonIoT.DescribeEndpointAsync(request);
            _logger.LogInformation($"Retrieved endpoint:
{response.EndpointAddress}");
            return response.EndpointAddress;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't describe endpoint. Here's why:
{ex.Message}");
            return null;
        }
    }

    /// <summary>
    /// Lists all certificates associated with the account.
    /// </summary>
```

```
    /// <returns>List of certificate information, or empty list if listing failed.</
returns>
    public async Task<List<Certificate>> ListCertificatesAsync()
    {
        try
        {
            var request = new ListCertificatesRequest();
            var response = await _amazonIoT.ListCertificatesAsync(request);

            _logger.LogInformation($"Retrieved {response.Certificates.Count}
 certificates");
            return response.Certificates;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
 {ex.Message}");
            return new List<Certificate>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't list certificates. Here's why:
 {ex.Message}");
            return new List<Certificate>();
        }
    }

    /// <summary>
    /// Updates the Thing's shadow with new state information.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
    /// <param name="shadowPayload">The shadow payload in JSON format.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> UpdateThingShadowAsync(string thingName, string
 shadowPayload)
    {
        try
        {
            var request = new UpdateThingShadowRequest
            {
                ThingName = thingName,
                Payload = new
 MemoryStream(System.Text.Encoding.UTF8.GetBytes(shadowPayload))
            };
```

```
            await _amazonIotData.UpdateThingShadowAsync(request);
            _logger.LogInformation($"Updated shadow for Thing {thingName}");
            return true;
        }
        catch (Amazon.IotData.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot update Thing shadow - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't update Thing shadow. Here's why:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Gets the Thing's shadow information.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
    /// <returns>The shadow data as a string, or null if retrieval failed.</returns>
    public async Task<string?> GetThingShadowAsync(string thingName)
    {
        try
        {
            var request = new GetThingShadowRequest
            {
                ThingName = thingName
            };

            var response = await _amazonIotData.GetThingShadowAsync(request);
            using var reader = new StreamReader(response.Payload);
            var shadowData = await reader.ReadToEndAsync();

            _logger.LogInformation($"Retrieved shadow for Thing {thingName}");
            return shadowData;
        }
        catch (Amazon.IotData.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot get Thing shadow - resource not found:
{ex.Message}");
```

```
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't get Thing shadow. Here's why:
{ex.Message}");
            return null;
        }
    }

    /// <summary>
    /// Creates an IoT topic rule.
    /// </summary>
    /// <param name="ruleName">The name of the rule.</param>
    /// <param name="snsTopicArn">The ARN of the SNS topic for the action.</param>
    /// <param name="roleArn">The ARN of the IAM role.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> CreateTopicRuleAsync(string ruleName, string
snsTopicArn, string roleArn)
    {
        try
        {
            var request = new CreateTopicRuleRequest
            {
                RuleName = ruleName,
                TopicRulePayload = new TopicRulePayload
                {
                    Sql = "SELECT * FROM 'topic/subtopic'",
                    Description = $"Rule created by .NET example: {ruleName}",
                    Actions = new List<Amazon.IoT.Model.Action>
                    {
                        new Amazon.IoT.Model.Action
                        {
                            Sns = new SnsAction
                            {
                                TargetArn = snsTopicArn,
                                RoleArn = roleArn
                            }
                        },
                    },
                    RuleDisabled = false
                }
            };
```

```
            await _amazonIoT.CreateTopicRuleAsync(request);
            _logger.LogInformation($"Created IoT rule {ruleName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceAlreadyExistsException ex)
        {
            _logger.LogWarning($"Rule {ruleName} already exists: {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't create topic rule. Here's why:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Deletes an IoT topic rule.
    /// </summary>
    /// <param name="ruleName">The name of the rule.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> DeleteTopicRuleAsync(string ruleName)
    {
        try
        {
            var request = new DeleteTopicRuleRequest
            {
                RuleName = ruleName,
            };

            await _amazonIoT.DeleteTopicRuleAsync(request);
            _logger.LogInformation($"Deleted IoT rule {ruleName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogWarning($"Rule {ruleName} not found: {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't delete topic rule. Here's why:
{ex.Message}");
```

```
            return false;
        }
    }

    /// <summary>
    /// Lists all IoT topic rules.
    /// </summary>
    /// <returns>List of topic rules, or empty list if listing failed.</returns>
    public async Task<List<TopicRuleListItem>> ListTopicRulesAsync()
    {
        try
        {
            var request = new ListTopicRulesRequest();
            var response = await _amazonIoT.ListTopicRulesAsync(request);

            _logger.LogInformation($"Retrieved {response.Rules.Count} IoT rules");
            return response.Rules;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return new List<TopicRuleListItem>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't list topic rules. Here's why:
{ex.Message}");
            return new List<TopicRuleListItem>();
        }
    }

    /// <summary>
    /// Searches for IoT Things using the search index.
    /// </summary>
    /// <param name="queryString">The search query string.</param>
    /// <returns>List of Things that match the search criteria, or empty list if
search failed.</returns>
    public async Task<List<ThingDocument>> SearchIndexAsync(string queryString)
    {
        try
        {
            // First, try to perform the search
            var request = new SearchIndexRequest
```

```
            {
                QueryString = queryString
            };

            var response = await _amazonIoT.SearchIndexAsync(request);
            _logger.LogInformation($"Search found {response.Things.Count} Things");
            return response.Things;
        }
        catch (Amazon.IoT.Model.IndexNotReadyException ex)
        {
            _logger.LogWarning($"Search index not ready, setting up indexing
configuration: {ex.Message}");
            return await SetupIndexAndRetrySearchAsync(queryString);
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex) when
(ex.Message.Contains("index") || ex.Message.Contains("Index"))
        {
            _logger.LogWarning($"Search index not configured, setting up indexing
configuration: {ex.Message}");
            return await SetupIndexAndRetrySearchAsync(queryString);
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return new List<ThingDocument>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't search index. Here's why: {ex.Message}");
            return new List<ThingDocument>();
        }
    }

    /// <summary>
    /// Sets up the indexing configuration and retries the search after waiting for
the index to be ready.
    /// </summary>
    /// <param name="queryString">The search query string.</param>
    /// <returns>List of Things that match the search criteria, or empty list if
setup/search failed.</returns>
    private async Task<List<ThingDocument>> SetupIndexAndRetrySearchAsync(string
queryString)
    {
```

```
        try
        {
            // Update indexing configuration to REGISTRY mode
            _logger.LogInformation("Setting up IoT search indexing
configuration...");
            await _amazonIoT.UpdateIndexingConfigurationAsync(
                new UpdateIndexingConfigurationRequest()
                {
                    ThingIndexingConfiguration = new ThingIndexingConfiguration()
                    {
                        ThingIndexingMode = ThingIndexingMode.REGISTRY
                    }
                });

            _logger.LogInformation("Indexing configuration updated. Waiting for
index to be ready...");

            // Wait for the index to be set up - this can take some time
            const int maxRetries = 10;
            const int retryDelaySeconds = 10;

            for (int attempt = 1; attempt <= maxRetries; attempt++)
            {
                try
                {
                    _logger.LogInformation($"Waiting for index to be ready (attempt
{attempt}/{maxRetries})...");
                    await Task.Delay(TimeSpan.FromSeconds(retryDelaySeconds));

                    // Try to get the current indexing configuration to see if it's
ready
                    var configResponse = await
_amazonIoT.GetIndexingConfigurationAsync(new GetIndexingConfigurationRequest());
                    if (configResponse.ThingIndexingConfiguration?.ThingIndexingMode
== ThingIndexingMode.REGISTRY)
                    {
                        // Try the search again
                        var request = new SearchIndexRequest
                        {
                            QueryString = queryString
                        };

                        var response = await _amazonIoT.SearchIndexAsync(request);
```

```
                        _logger.LogInformation($"Search found
{response.Things.Count} Things after index setup");
                        return response.Things;
                    }
                }
                catch (Amazon.IoT.Model.IndexNotReadyException)
                {
                    // Index still not ready, continue waiting
                    _logger.LogInformation("Index still not ready, continuing to
wait...");
                    continue;
                }
                catch (Amazon.IoT.Model.InvalidRequestException ex) when
(ex.Message.Contains("index") || ex.Message.Contains("Index"))
                {
                    // Index still not ready, continue waiting
                    _logger.LogInformation("Index still not ready, continuing to
wait...");
                    continue;
                }
            }

            _logger.LogWarning("Timeout waiting for search index to be ready after
configuration update");
            return new List<ThingDocument>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't set up search index configuration. Here's
why: {ex.Message}");
            return new List<ThingDocument>();
        }
    }

    /// <summary>
    /// Detaches a certificate from an IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
    /// <param name="certificateArn">The ARN of the certificate to detach.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> DetachThingPrincipalAsync(string thingName, string
certificateArn)
    {
        try
```

```
        {
            var request = new DetachThingPrincipalRequest
            {
                ThingName = thingName,
                Principal = certificateArn
            };

            await _amazonIoT.DetachThingPrincipalAsync(request);
            _logger.LogInformation($"Detached certificate {certificateArn} from
Thing {thingName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot detach certificate - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't detach certificate from Thing. Here's why:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Deletes an IoT certificate.
    /// </summary>
    /// <param name="certificateId">The ID of the certificate to delete.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> DeleteCertificateAsync(string certificateId)
    {
        try
        {
            // First, update the certificate to inactive state
            var updateRequest = new UpdateCertificateRequest
            {
                CertificateId = certificateId,
                NewStatus = CertificateStatus.INACTIVE
            };
            await _amazonIoT.UpdateCertificateAsync(updateRequest);

            // Then delete the certificate
```

```
            var deleteRequest = new DeleteCertificateRequest
            {
                CertificateId = certificateId
            };

            await _amazonIoT.DeleteCertificateAsync(deleteRequest);
            _logger.LogInformation($"Deleted certificate {certificateId}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot delete certificate - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't delete certificate. Here's why:
{ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Deletes an IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing to delete.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> DeleteThingAsync(string thingName)
    {
        try
        {
            var request = new DeleteThingRequest
            {
                ThingName = thingName
            };

            await _amazonIoT.DeleteThingAsync(request);
            _logger.LogInformation($"Deleted Thing {thingName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
```

```
            _logger.LogError($"Cannot delete Thing - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't delete Thing. Here's why: {ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Lists IoT Things with pagination support.
    /// </summary>
    /// <returns>List of Things, or empty list if listing failed.</returns>
    public async Task<List<ThingAttribute>> ListThingsAsync()
    {
        try
        {
            // Use pages of 10.
            var request = new ListThingsRequest()
            {
                MaxResults = 10
            };
            var response = await _amazonIoT.ListThingsAsync(request);

            // Since there is not a built-in paginator, use the NextMarker to
paginate.
            bool hasMoreResults = true;

            var things = new List<ThingAttribute>();
            while (hasMoreResults)
            {
                things.AddRange(response.Things);

                // If NextMarker is not null, there are more results. Get the next
page of results.
                if (!String.IsNullOrEmpty(response.NextMarker))
                {
                    request.Marker = response.NextMarker;
                    response = await _amazonIoT.ListThingsAsync(request);
                }
                else
                    hasMoreResults = false;
```

```
            }

            _logger.LogInformation($"Retrieved {things.Count} Things");
            return things;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
 {ex.Message}");
            return new List<ThingAttribute>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't list Things. Here's why: {ex.Message}");
            return new List<ThingAttribute>();
        }
    }

}
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para .NET .
  - AttachThingPrincipal
  - CreateKeysAndCertificate
  - CreateThing
  - CreateTopicRule
  - DeleteCertificate
  - DeleteThing
  - DeleteTopicRule
  - DescribeEndpoint
  - DescribeThing
  - DetachThingPrincipal
  - ListCertificates
  - ListThings
  - SearchIndex
  - UpdateIndexingConfiguration

- UpdateThing

Ações

`AttachThingPrincipal`

O código de exemplo a seguir mostra como usar `AttachThingPrincipal`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
   /// <summary>
   /// Attaches a certificate to an IoT Thing.
   /// </summary>
   /// <param name="thingName">The name of the Thing.</param>
   /// <param name="certificateArn">The ARN of the certificate to attach.</param>
   /// <returns>True if successful, false otherwise.</returns>
   public async Task<bool> AttachThingPrincipalAsync(string thingName, string
certificateArn)
   {
       try
       {
           var request = new AttachThingPrincipalRequest
           {
               ThingName = thingName,
               Principal = certificateArn
           };

           await _amazonIoT.AttachThingPrincipalAsync(request);
           _logger.LogInformation($"Attached certificate {certificateArn} to Thing
{thingName}");
           return true;
       }
       catch (Amazon.IoT.Model.ResourceNotFoundException ex)
       {
```

```
            _logger.LogError($"Cannot attach certificate - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't attach certificate to Thing. Here's why:
{ex.Message}");
            return false;
        }
    }
```

- Para obter detalhes da API, consulte AttachThingPrincipala Referência AWS SDK para .NET da API.

`CreateKeysAndCertificate`

O código de exemplo a seguir mostra como usar `CreateKeysAndCertificate`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Creates a device certificate for AWS IoT.
    /// </summary>
    /// <returns>The certificate details including ARN and certificate PEM, or null
if creation failed.</returns>
    public async Task<(string CertificateArn, string CertificatePem, string
CertificateId)?> CreateKeysAndCertificateAsync()
    {
        try
        {
            var request = new CreateKeysAndCertificateRequest
            {
                SetAsActive = true
```

```
            };

            var response = await _amazonIoT.CreateKeysAndCertificateAsync(request);
            _logger.LogInformation($"Created certificate with ARN
{response.CertificateArn}");
            return (response.CertificateArn, response.CertificatePem,
response.CertificateId);
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't create certificate. Here's why:
{ex.Message}");
            return null;
        }
    }
```

- Para obter detalhes da API, consulte CreateKeysAndCertificatea Referência AWS SDK para .NET da API.

`CreateThing`

O código de exemplo a seguir mostra como usar `CreateThing`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Creates an AWS IoT Thing.
    /// </summary>
```

```
    /// <param name="thingName">The name of the Thing to create.</param>
    /// <returns>The ARN of the Thing created, or null if creation failed.</returns>
    public async Task<string?> CreateThingAsync(string thingName)
    {
        try
        {
            var request = new CreateThingRequest
            {
                ThingName = thingName
            };

            var response = await _amazonIoT.CreateThingAsync(request);
            _logger.LogInformation($"Created Thing {thingName} with ARN
{response.ThingArn}");
            return response.ThingArn;
        }
        catch (Amazon.IoT.Model.ResourceAlreadyExistsException ex)
        {
            _logger.LogWarning($"Thing {thingName} already exists: {ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't create Thing {thingName}. Here's why:
{ex.Message}");
            return null;
        }
    }
```

- Para obter detalhes da API, consulte CreateThinga Referência AWS SDK para .NET da API.

`CreateTopicRule`

O código de exemplo a seguir mostra como usar `CreateTopicRule`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Creates an IoT topic rule.
    /// </summary>
    /// <param name="ruleName">The name of the rule.</param>
    /// <param name="snsTopicArn">The ARN of the SNS topic for the action.</param>
    /// <param name="roleArn">The ARN of the IAM role.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> CreateTopicRuleAsync(string ruleName, string
snsTopicArn, string roleArn)
    {
        try
        {
            var request = new CreateTopicRuleRequest
            {
                RuleName = ruleName,
                TopicRulePayload = new TopicRulePayload
                {
                    Sql = "SELECT * FROM 'topic/subtopic'",
                    Description = $"Rule created by .NET example: {ruleName}",
                    Actions = new List<Amazon.IoT.Model.Action>
                    {
                        new Amazon.IoT.Model.Action
                        {
                            Sns = new SnsAction
                            {
                                TargetArn = snsTopicArn,
                                RoleArn = roleArn
                            }
                        },
                        RuleDisabled = false
                    }
            };

            await _amazonIoT.CreateTopicRuleAsync(request);
            _logger.LogInformation($"Created IoT rule {ruleName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceAlreadyExistsException ex)
        {
            _logger.LogWarning($"Rule {ruleName} already exists: {ex.Message}");
            return false;
        }
```

```
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't create topic rule. Here's why:
{ex.Message}");
            return false;
        }
    }
```

- Para obter detalhes da API, consulte CreateTopicRulea Referência AWS SDK para .NET da API.

`DeleteCertificate`

O código de exemplo a seguir mostra como usar `DeleteCertificate`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Deletes an IoT certificate.
    /// </summary>
    /// <param name="certificateId">The ID of the certificate to delete.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> DeleteCertificateAsync(string certificateId)
    {
        try
        {
            // First, update the certificate to inactive state
            var updateRequest = new UpdateCertificateRequest
            {
                CertificateId = certificateId,
                NewStatus = CertificateStatus.INACTIVE
            };
            await _amazonIoT.UpdateCertificateAsync(updateRequest);
```

```
            // Then delete the certificate
            var deleteRequest = new DeleteCertificateRequest
            {
                CertificateId = certificateId
            };

            await _amazonIoT.DeleteCertificateAsync(deleteRequest);
            _logger.LogInformation($"Deleted certificate {certificateId}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot delete certificate - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't delete certificate. Here's why:
{ex.Message}");
            return false;
        }
    }
```

- Para obter detalhes da API, consulte DeleteCertificatea Referência AWS SDK para .NET da API.

`DeleteThing`

O código de exemplo a seguir mostra como usar `DeleteThing`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
```

```
    /// Deletes an IoT Thing.
    /// </summary>
    /// <param name="thingName">The name of the Thing to delete.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> DeleteThingAsync(string thingName)
    {
        try
        {
            var request = new DeleteThingRequest
            {
                ThingName = thingName
            };

            await _amazonIoT.DeleteThingAsync(request);
            _logger.LogInformation($"Deleted Thing {thingName}");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot delete Thing - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't delete Thing. Here's why: {ex.Message}");
            return false;
        }
    }
```

- Para obter detalhes da API, consulte DeleteThinga Referência AWS SDK para .NET da API.

`DescribeEndpoint`

O código de exemplo a seguir mostra como usar `DescribeEndpoint`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Gets the AWS IoT endpoint URL.
    /// </summary>
    /// <returns>The endpoint URL, or null if retrieval failed.</returns>
    public async Task<string?> DescribeEndpointAsync()
    {
        try
        {
            var request = new DescribeEndpointRequest
            {
                EndpointType = "iot:Data-ATS"
            };

            var response = await _amazonIoT.DescribeEndpointAsync(request);
            _logger.LogInformation($"Retrieved endpoint:
{response.EndpointAddress}");
            return response.EndpointAddress;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return null;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't describe endpoint. Here's why:
{ex.Message}");
            return null;
        }
    }
```

- Para obter detalhes da API, consulte DescribeEndpointa Referência AWS SDK para .NET da API.

`DetachThingPrincipal`

O código de exemplo a seguir mostra como usar `DetachThingPrincipal`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
   /// <summary>
   /// Detaches a certificate from an IoT Thing.
   /// </summary>
   /// <param name="thingName">The name of the Thing.</param>
   /// <param name="certificateArn">The ARN of the certificate to detach.</param>
   /// <returns>True if successful, false otherwise.</returns>
   public async Task<bool> DetachThingPrincipalAsync(string thingName, string
certificateArn)
   {
       try
       {
           var request = new DetachThingPrincipalRequest
           {
               ThingName = thingName,
               Principal = certificateArn
           };

           await _amazonIoT.DetachThingPrincipalAsync(request);
           _logger.LogInformation($"Detached certificate {certificateArn} from
Thing {thingName}");
           return true;
       }
       catch (Amazon.IoT.Model.ResourceNotFoundException ex)
       {
           _logger.LogError($"Cannot detach certificate - resource not found:
{ex.Message}");
           return false;
```

```
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't detach certificate from Thing. Here's why:
 {ex.Message}");
            return false;
        }
    }
```

- Para obter detalhes da API, consulte DetachThingPrincipala Referência AWS SDK para .NET da API.

`ListCertificates`

O código de exemplo a seguir mostra como usar `ListCertificates`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Lists all certificates associated with the account.
    /// </summary>
    /// <returns>List of certificate information, or empty list if listing failed.</
returns>
    public async Task<List<Certificate>> ListCertificatesAsync()
    {
        try
        {
            var request = new ListCertificatesRequest();
            var response = await _amazonIoT.ListCertificatesAsync(request);

            _logger.LogInformation($"Retrieved {response.Certificates.Count}
 certificates");
            return response.Certificates;
        }
```

```
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return new List<Certificate>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't list certificates. Here's why:
{ex.Message}");
            return new List<Certificate>();
        }
    }
```

- Para obter detalhes da API, consulte ListCertificatesa Referência AWS SDK para .NET da API.

`ListThings`

O código de exemplo a seguir mostra como usar `ListThings`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Lists IoT Things with pagination support.
    /// </summary>
    /// <returns>List of Things, or empty list if listing failed.</returns>
    public async Task<List<ThingAttribute>> ListThingsAsync()
    {
        try
        {
            // Use pages of 10.
            var request = new ListThingsRequest()
            {
                MaxResults = 10
```

```
            };
            var response = await _amazonIoT.ListThingsAsync(request);

            // Since there is not a built-in paginator, use the NextMarker to
paginate.
            bool hasMoreResults = true;

            var things = new List<ThingAttribute>();
            while (hasMoreResults)
            {
                things.AddRange(response.Things);

                // If NextMarker is not null, there are more results. Get the next
page of results.
                if (!String.IsNullOrEmpty(response.NextMarker))
                {
                    request.Marker = response.NextMarker;
                    response = await _amazonIoT.ListThingsAsync(request);
                }
                else
                    hasMoreResults = false;
            }

            _logger.LogInformation($"Retrieved {things.Count} Things");
            return things;
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return new List<ThingAttribute>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't list Things. Here's why: {ex.Message}");
            return new List<ThingAttribute>();
        }
    }
```

- Para obter detalhes da API, consulte ListThingsa Referência AWS SDK para .NET da API.

`SearchIndex`

O código de exemplo a seguir mostra como usar `SearchIndex`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
   /// <summary>
   /// Searches for IoT Things using the search index.
   /// </summary>
   /// <param name="queryString">The search query string.</param>
   /// <returns>List of Things that match the search criteria, or empty list if
search failed.</returns>
   public async Task<List<ThingDocument>> SearchIndexAsync(string queryString)
   {
       try
       {
           // First, try to perform the search
           var request = new SearchIndexRequest
           {
               QueryString = queryString
           };

           var response = await _amazonIoT.SearchIndexAsync(request);
           _logger.LogInformation($"Search found {response.Things.Count} Things");
           return response.Things;
       }
       catch (Amazon.IoT.Model.IndexNotReadyException ex)
       {
           _logger.LogWarning($"Search index not ready, setting up indexing
configuration: {ex.Message}");
           return await SetupIndexAndRetrySearchAsync(queryString);
       }
       catch (Amazon.IoT.Model.ResourceNotFoundException ex) when
(ex.Message.Contains("index") || ex.Message.Contains("Index"))
       {
           _logger.LogWarning($"Search index not configured, setting up indexing
configuration: {ex.Message}");
```

```
            return await SetupIndexAndRetrySearchAsync(queryString);
        }
        catch (Amazon.IoT.Model.ThrottlingException ex)
        {
            _logger.LogWarning($"Request throttled, please try again later:
{ex.Message}");
            return new List<ThingDocument>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't search index. Here's why: {ex.Message}");
            return new List<ThingDocument>();
        }
    }

    /// <summary>
    /// Sets up the indexing configuration and retries the search after waiting for
the index to be ready.
    /// </summary>
    /// <param name="queryString">The search query string.</param>
    /// <returns>List of Things that match the search criteria, or empty list if
setup/search failed.</returns>
    private async Task<List<ThingDocument>> SetupIndexAndRetrySearchAsync(string
queryString)
    {
        try
        {
            // Update indexing configuration to REGISTRY mode
            _logger.LogInformation("Setting up IoT search indexing
configuration...");
            await _amazonIoT.UpdateIndexingConfigurationAsync(
                new UpdateIndexingConfigurationRequest()
                {
                    ThingIndexingConfiguration = new ThingIndexingConfiguration()
                    {
                        ThingIndexingMode = ThingIndexingMode.REGISTRY
                    }
                });

            _logger.LogInformation("Indexing configuration updated. Waiting for
index to be ready...");

            // Wait for the index to be set up - this can take some time
            const int maxRetries = 10;
```

```
            const int retryDelaySeconds = 10;

            for (int attempt = 1; attempt <= maxRetries; attempt++)
            {
                try
                {
                    _logger.LogInformation($"Waiting for index to be ready (attempt
{attempt}/{maxRetries})...");
                    await Task.Delay(TimeSpan.FromSeconds(retryDelaySeconds));

                    // Try to get the current indexing configuration to see if it's
ready
                    var configResponse = await
_amazonIoT.GetIndexingConfigurationAsync(new GetIndexingConfigurationRequest());
                    if (configResponse.ThingIndexingConfiguration?.ThingIndexingMode
== ThingIndexingMode.REGISTRY)
                    {
                        // Try the search again
                        var request = new SearchIndexRequest
                        {
                            QueryString = queryString
                        };

                        var response = await _amazonIoT.SearchIndexAsync(request);
                        _logger.LogInformation($"Search found
{response.Things.Count} Things after index setup");
                        return response.Things;
                    }
                }
                catch (Amazon.IoT.Model.IndexNotReadyException)
                {
                    // Index still not ready, continue waiting
                    _logger.LogInformation("Index still not ready, continuing to
wait...");
                    continue;
                }
                catch (Amazon.IoT.Model.InvalidRequestException ex) when
(ex.Message.Contains("index") || ex.Message.Contains("Index"))
                {
                    // Index still not ready, continue waiting
                    _logger.LogInformation("Index still not ready, continuing to
wait...");
                    continue;
                }
```

```
            }

            _logger.LogWarning("Timeout waiting for search index to be ready after
configuration update");
            return new List<ThingDocument>();
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't set up search index configuration. Here's
why: {ex.Message}");
            return new List<ThingDocument>();
        }
    }
```

- Para obter detalhes da API, consulte SearchIndexa Referência AWS SDK para .NET da API.

`UpdateThing`

O código de exemplo a seguir mostra como usar `UpdateThing`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Updates an IoT Thing with attributes.
    /// </summary>
    /// <param name="thingName">The name of the Thing to update.</param>
    /// <param name="attributes">Dictionary of attributes to add.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> UpdateThingAsync(string thingName, Dictionary<string,
string> attributes)
    {
        try
        {
            var request = new UpdateThingRequest
            {
```

```
                ThingName = thingName,
                AttributePayload = new AttributePayload
                {
                    Attributes = attributes,
                    Merge = true
                }
            };

            await _amazonIoT.UpdateThingAsync(request);
            _logger.LogInformation($"Updated Thing {thingName} with attributes");
            return true;
        }
        catch (Amazon.IoT.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot update Thing - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't update Thing attributes. Here's why:
{ex.Message}");
            return false;
        }
    }
```

- Para obter detalhes da API, consulte UpdateThinga Referência AWS SDK para .NET da API.

AWS IoT data exemplos usando SDK para .NET (v4)

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET (v4) com AWS IoT data.

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar perfis de serviço individuais, você pode ver as ações no contexto em seus cenários relacionados.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- Ações

Ações

`GetThingShadow`

O código de exemplo a seguir mostra como usar `GetThingShadow`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Gets the Thing's shadow information.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
    /// <returns>The shadow data as a string, or null if retrieval failed.</returns>
    public async Task<string?> GetThingShadowAsync(string thingName)
    {
        try
        {
            var request = new GetThingShadowRequest
            {
                ThingName = thingName
            };

            var response = await _amazonIotData.GetThingShadowAsync(request);
            using var reader = new StreamReader(response.Payload);
            var shadowData = await reader.ReadToEndAsync();

            _logger.LogInformation($"Retrieved shadow for Thing {thingName}");
            return shadowData;
        }
        catch (Amazon.IotData.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot get Thing shadow - resource not found:
{ex.Message}");
            return null;
```

```
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't get Thing shadow. Here's why:
{ex.Message}");
            return null;
        }
    }
```

- Para obter detalhes da API, consulte GetThingShadowa Referência AWS SDK para .NET da API.

`UpdateThingShadow`

O código de exemplo a seguir mostra como usar `UpdateThingShadow`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Updates the Thing's shadow with new state information.
    /// </summary>
    /// <param name="thingName">The name of the Thing.</param>
    /// <param name="shadowPayload">The shadow payload in JSON format.</param>
    /// <returns>True if successful, false otherwise.</returns>
    public async Task<bool> UpdateThingShadowAsync(string thingName, string
shadowPayload)
    {
        try
        {
            var request = new UpdateThingShadowRequest
            {
                ThingName = thingName,
                Payload = new
MemoryStream(System.Text.Encoding.UTF8.GetBytes(shadowPayload))
```

```
            };

            await _amazonIotData.UpdateThingShadowAsync(request);
            _logger.LogInformation($"Updated shadow for Thing {thingName}");
            return true;
        }
        catch (Amazon.IotData.Model.ResourceNotFoundException ex)
        {
            _logger.LogError($"Cannot update Thing shadow - resource not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            _logger.LogError($"Couldn't update Thing shadow. Here's why:
{ex.Message}");
            return false;
        }
    }
```

- Para obter detalhes da API, consulte UpdateThingShadowa Referência AWS SDK para .NET da API.

Exemplos do Amazon Redshift usando SDK para .NET (v4)

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET (v4) com o Amazon Redshift.

As noções básicas são exemplos de código que mostram como realizar as operações essenciais em um serviço.

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar perfis de serviço individuais, você pode ver as ações no contexto em seus cenários relacionados.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- Conceitos básicos

- Conceitos básicos
- Ações

Conceitos básicos

Olá, Amazon Redshift

O exemplo de código a seguir mostra como começar a usar o Amazon Redshift.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Main method to run the Hello Amazon Redshift example.
    /// </summary>
    /// <param name="args">Command line arguments (not used).</param>
    public static async Task Main(string[] args)
    {
        var redshiftClient = new AmazonRedshiftClient();

        Console.WriteLine("Hello, Amazon Redshift! Let's list available clusters:");

        var clusters = new List<Cluster>();

        try
        {
            // Use pagination to retrieve all clusters.
            var clustersPaginator = redshiftClient.Paginators.DescribeClusters(new
DescribeClustersRequest());

            await foreach (var response in clustersPaginator.Responses)
            {
                if (response.Clusters != null)
                    clusters.AddRange(response.Clusters);
            }
```

```
            Console.WriteLine($"{clusters.Count} cluster(s) retrieved.");

            foreach (var cluster in clusters)
            {
                Console.WriteLine($"\t{cluster.ClusterIdentifier} (Status:
{cluster.ClusterStatus})");
            }
        }
        catch (AmazonRedshiftException ex)
        {
            Console.WriteLine($"Couldn't list clusters. Here's why: {ex.Message}");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred: {ex.Message}");
        }
    }
```

- Para obter detalhes da API, consulte DescribeClusters a Referência AWS SDK para .NET da API.

Conceitos básicos

Conheça os conceitos básicos

O exemplo de código a seguir mostra como:

- Criar um cluster do Redshift.
- Listar bancos de dados no cluster.
- Criar uma tabela chamada Filmes.
- Preencher a tabela Filmes.
- Consultar a tabela Filmes por ano.
- Modificar o cluster do Redshift.
- Excluir o cluster do Amazon Redshift.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Crie uma classe de wrapper do Redshift para gerenciar as operações.

```
/// <summary>
/// Wrapper class for Amazon Redshift operations.
/// </summary>
public class RedshiftWrapper
{
    private readonly IAmazonRedshift _redshiftClient;
    private readonly IAmazonRedshiftDataAPIService _redshiftDataClient;

    /// <summary>
    /// Constructor for RedshiftWrapper.
    /// </summary>
    /// <param name="redshiftClient">Amazon Redshift client.</param>
    /// <param name="redshiftDataClient">Amazon Redshift Data API client.</param>
    public RedshiftWrapper(IAmazonRedshift redshiftClient,
 IAmazonRedshiftDataAPIService redshiftDataClient)
    {
        _redshiftClient = redshiftClient;
        _redshiftDataClient = redshiftDataClient;
    }

    /// <summary>
    /// Create a new Amazon Redshift cluster.
    /// </summary>
    /// <param name="clusterIdentifier">The identifier for the cluster.</param>
    /// <param name="databaseName">The name of the database.</param>
    /// <param name="masterUsername">The master username.</param>
    /// <param name="masterUserPassword">The master user password.</param>
    /// <param name="nodeType">The node type for the cluster.</param>
    /// <returns>The cluster that was created.</returns>
    public async Task<Cluster> CreateClusterAsync(string clusterIdentifier, string
 databaseName,
        string masterUsername, string masterUserPassword, string nodeType =
 "ra3.large")
```

```
    {
        try
        {
            var request = new CreateClusterRequest
            {
                ClusterIdentifier = clusterIdentifier,
                DBName = databaseName,
                MasterUsername = masterUsername,
                MasterUserPassword = masterUserPassword,
                NodeType = nodeType,
                NumberOfNodes = 1,
                ClusterType = "single-node"
            };

            var response = await _redshiftClient.CreateClusterAsync(request);
            Console.WriteLine($"Created cluster {clusterIdentifier}");
            return response.Cluster;
        }
        catch (ClusterAlreadyExistsException ex)
        {
            Console.WriteLine($"Cluster already exists: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't create cluster. Here's why: {ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Describe Amazon Redshift clusters.
    /// </summary>
    /// <param name="clusterIdentifier">Optional cluster identifier to describe a
specific cluster.</param>
    /// <returns>A list of clusters.</returns>
    public async Task<List<Cluster>> DescribeClustersAsync(string? clusterIdentifier
= null)
    {
        try
        {
            var clusters = new List<Cluster>();
            var request = new DescribeClustersRequest();
            if (!string.IsNullOrEmpty(clusterIdentifier))
```

```
            {
                request.ClusterIdentifier = clusterIdentifier;
            }

            var clustersPaginator =
 _redshiftClient.Paginators.DescribeClusters(request);
            await foreach (var response in clustersPaginator.Responses)
            {
                if (response.Clusters != null)
                    clusters.AddRange(response.Clusters);
            }

            Console.WriteLine($"{clusters.Count} cluster(s) retrieved.");
            foreach (var cluster in clusters)
            {
                Console.WriteLine($"\t{cluster.ClusterIdentifier} (Status:
 {cluster.ClusterStatus})");
            }

            return clusters;
        }
        catch (ClusterNotFoundException ex)
        {
            Console.WriteLine($"Cluster {clusterIdentifier} not found:
 {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't describe clusters. Here's why:
 {ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Modify an Amazon Redshift cluster.
    /// </summary>
    /// <param name="clusterIdentifier">The identifier for the cluster.</param>
    /// <param name="preferredMaintenanceWindow">The preferred maintenance window.</
param>
    /// <returns>True if successful.</returns>
    public async Task<bool> ModifyClusterAsync(string clusterIdentifier, string
 preferredMaintenanceWindow)
```

```
    {
        try
        {
            var request = new ModifyClusterRequest
            {
                ClusterIdentifier = clusterIdentifier,
                PreferredMaintenanceWindow = preferredMaintenanceWindow
            };

            var response = await _redshiftClient.ModifyClusterAsync(request);
            Console.WriteLine($"The modified cluster was successfully modified and
has {response.Cluster.PreferredMaintenanceWindow} as the maintenance window");
            return true;
        }
        catch (ClusterNotFoundException ex)
        {
            Console.WriteLine($"Cluster {clusterIdentifier} not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't modify cluster. Here's why: {ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// Delete an Amazon Redshift cluster without a final snapshot.
    /// </summary>
    /// <param name="clusterIdentifier">The identifier for the cluster.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> DeleteClusterWithoutSnapshotAsync(string
clusterIdentifier)
    {
        try
        {
            var request = new DeleteClusterRequest
            {
                ClusterIdentifier = clusterIdentifier,
                SkipFinalClusterSnapshot = true
            };

            var response = await _redshiftClient.DeleteClusterAsync(request);
```

```
            Console.WriteLine($"The {clusterIdentifier} was deleted");
            return true;
        }
        catch (ClusterNotFoundException ex)
        {
            Console.WriteLine($"Cluster not found: {ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't delete cluster. Here's why: {ex.Message}");
            return false;
        }
    }

    /// <summary>
    /// List databases in a Redshift cluster.
    /// </summary>
    /// <param name="clusterIdentifier">The cluster identifier.</param>
    /// <param name="dbUser">The database user.</param>
    /// <param name="dbUser">The database name for authentication.</param>
    /// <returns>A list of database names.</returns>
    public async Task<List<string>> ListDatabasesAsync(string clusterIdentifier,
string dbUser, string databaseName)
    {
        try
        {
            var request = new ListDatabasesRequest
            {
                ClusterIdentifier = clusterIdentifier,
                DbUser = dbUser,
                Database = databaseName
            };

            var response = await _redshiftDataClient.ListDatabasesAsync(request);
            var databases = new List<string>();

            foreach (var database in response.Databases)
            {
                Console.WriteLine($"The database name is : {database}");
                databases.Add(database);
            }

            return databases;
```

```
        }
        catch (Amazon.RedshiftDataAPIService.Model.ValidationException ex)
        {
            Console.WriteLine($"Validation error: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't list databases. Here's why: {ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Create a table in the Redshift database.
    /// </summary>
    /// <param name="clusterIdentifier">The cluster identifier.</param>
    /// <param name="database">The database name.</param>
    /// <param name="dbUser">The database user.</param>
    /// <returns>The statement ID.</returns>
    public async Task<string> CreateTableAsync(string clusterIdentifier, string
database, string dbUser)
    {
        try
        {
            var sqlStatement = @"
                CREATE TABLE Movies (
                    id INTEGER PRIMARY KEY,
                    title VARCHAR(250) NOT NULL,
                    year INTEGER NOT NULL
                )";

            var request = new ExecuteStatementRequest
            {
                ClusterIdentifier = clusterIdentifier,
                Database = database,
                DbUser = dbUser,
                Sql = sqlStatement
            };

            var response = await _redshiftDataClient.ExecuteStatementAsync(request);
            await WaitForStatementToCompleteAsync(response.Id);
            Console.WriteLine("Table created: Movies");
            return response.Id;
```

```
        }
        catch (Amazon.RedshiftDataAPIService.Model.ValidationException ex)
        {
            Console.WriteLine($"Validation error: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't create table. Here's why: {ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Insert a record into the Movies table using parameterized query.
    /// </summary>
    /// <param name="clusterIdentifier">The cluster identifier.</param>
    /// <param name="database">The database name.</param>
    /// <param name="dbUser">The database user.</param>
    /// <param name="id">The movie ID.</param>
    /// <param name="title">The movie title.</param>
    /// <param name="year">The movie year.</param>
    /// <returns>The statement ID.</returns>
    public async Task<string> InsertMovieAsync(string clusterIdentifier, string
database, string dbUser,
        int id, string title, int year)
    {
        try
        {
            var sqlStatement = "INSERT INTO Movies (id, title, year) VALUES
(:id, :title, :year)";

            var request = new ExecuteStatementRequest
            {
                ClusterIdentifier = clusterIdentifier,
                Database = database,
                DbUser = dbUser,
                Sql = sqlStatement,
                Parameters = new List<SqlParameter>
                {
                    new SqlParameter { Name = "id", Value = id.ToString() },
                    new SqlParameter { Name = "title", Value = title },
                    new SqlParameter { Name = "year", Value = year.ToString() }
                }
```

```
            };

            var response = await _redshiftDataClient.ExecuteStatementAsync(request);
            await WaitForStatementToCompleteAsync(response.Id);
            Console.WriteLine($"Inserted: {title} ({year})");
            return response.Id;
        }
        catch (Amazon.RedshiftDataAPIService.Model.ValidationException ex)
        {
            Console.WriteLine($"Validation error: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't insert movie. Here's why: {ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Query movies by year using parameterized query.
    /// </summary>
    /// <param name="clusterIdentifier">The cluster identifier.</param>
    /// <param name="database">The database name.</param>
    /// <param name="dbUser">The database user.</param>
    /// <param name="year">The year to query.</param>
    /// <returns>A list of movie titles.</returns>
    public async Task<List<string>> QueryMoviesByYearAsync(string clusterIdentifier,
string database,
        string dbUser, int year)
    {
        try
        {
            var sqlStatement = "SELECT title FROM Movies WHERE year = :year";

            var request = new ExecuteStatementRequest
            {
                ClusterIdentifier = clusterIdentifier,
                Database = database,
                DbUser = dbUser,
                Sql = sqlStatement,
                Parameters = new List<SqlParameter>
                {
                    new SqlParameter { Name = "year", Value = year.ToString() }
```

```
                }
            };

            var response = await _redshiftDataClient.ExecuteStatementAsync(request);
            Console.WriteLine($"The identifier of the statement is {response.Id}");

            await WaitForStatementToCompleteAsync(response.Id);

            var results = await GetStatementResultAsync(response.Id);
            var movieTitles = new List<string>();

            foreach (var row in results)
            {
                if (row.Count > 0)
                {
                    var title = row[0].StringValue;
                    Console.WriteLine($"The Movie title field is {title}");
                    movieTitles.Add(title);
                }
            }

            return movieTitles;
        }
        catch (Amazon.RedshiftDataAPIService.Model.ValidationException ex)
        {
            Console.WriteLine($"Validation error: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't query movies. Here's why: {ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Describe a statement execution.
    /// </summary>
    /// <param name="statementId">The statement ID.</param>
    /// <returns>The statement description.</returns>
    public async Task<DescribeStatementResponse> DescribeStatementAsync(string
statementId)
    {
        try
```

```
        {
            var request = new DescribeStatementRequest
            {
                Id = statementId
            };

            var response = await
_redshiftDataClient.DescribeStatementAsync(request);
            return response;
        }
        catch (Amazon.RedshiftDataAPIService.Model.ResourceNotFoundException ex)
        {
            Console.WriteLine($"Statement not found: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't describe statement. Here's why:
{ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Get the results of a statement execution.
    /// </summary>
    /// <param name="statementId">The statement ID.</param>
    /// <returns>A list of result rows.</returns>
    public async Task<List<List<Field>>> GetStatementResultAsync(string statementId)
    {
        try
        {
            var request = new GetStatementResultRequest
            {
                Id = statementId
            };

            var response = await
_redshiftDataClient.GetStatementResultAsync(request);
            return response.Records;
        }
        catch (Amazon.RedshiftDataAPIService.Model.ResourceNotFoundException ex)
        {
            Console.WriteLine($"Statement not found: {ex.Message}");
```

```
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't get statement result. Here's why:
{ex.Message}");
            throw;
        }
    }

    /// <summary>
    /// Wait for a statement to complete execution.
    /// </summary>
    /// <param name="statementId">The statement ID.</param>
    /// <returns>A task representing the asynchronous operation.</returns>
    private async Task WaitForStatementToCompleteAsync(string statementId)
    {
        var status = StatusString.SUBMITTED;
        DescribeStatementResponse? response = null;

        while (status == StatusString.SUBMITTED || status == StatusString.PICKED ||
status == StatusString.STARTED)
        {
            await Task.Delay(1000); // Wait 1 second
            response = await DescribeStatementAsync(statementId);
            status = response.Status;
            Console.WriteLine($"...{status}");
        }

        if (status == StatusString.FINISHED)
        {
            Console.WriteLine("The statement is finished!");
        }
        else
        {
            var errorMessage = response?.Error ?? "Unknown error";
            Console.WriteLine($"The statement failed with status: {status}");
            Console.WriteLine($"Error message: {errorMessage}");
        }
    }

    /// <summary>
    /// Wait for a cluster to become available.
    /// </summary>
```

```
    /// <param name="clusterIdentifier">The cluster identifier.</param>
    /// <returns>A task representing the asynchronous operation.</returns>
    public async Task WaitForClusterAvailableAsync(string clusterIdentifier)
    {
        Console.WriteLine($"Wait until {clusterIdentifier} is available. This may
 take a few minutes.");

        var startTime = DateTime.Now;
        var clusters = await DescribeClustersAsync(clusterIdentifier);

        while (clusters[0].ClusterStatus != "available")
        {
            var elapsed = DateTime.Now - startTime;
            Console.WriteLine($"Elapsed Time: {elapsed:mm\\:ss} - Waiting for
 cluster...");

            await Task.Delay(5000); // Wait 5 seconds
            clusters = await DescribeClustersAsync(clusterIdentifier);
        }

        var totalElapsed = DateTime.Now - startTime;
        Console.WriteLine($"Cluster is available! Total Elapsed Time:
 {totalElapsed:mm\\:ss}");
    }
}
```

Execute um cenário interativo que demonstra os fundamentos do Redshift.

```
/// <summary>
/// Amazon Redshift Getting Started Scenario.
/// </summary>
public class RedshiftBasics
{
    public static bool IsInteractive = true;
    public static RedshiftWrapper? Wrapper = null;
    public static ILogger logger = null!;
    private static readonly string _moviesFilePath = "../../../../../../resources/
sample_files/movies.json";

    /// <summary>
    /// Main method for the Amazon Redshift Getting Started scenario.
    /// </summary>
```

```
    /// <param name="args">Command line arguments.</param>
    public static async Task Main(string[] args)
    {
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonRedshift>()
                    .AddAWSService<IAmazonRedshiftDataAPIService>()
                    .AddTransient<RedshiftWrapper>()
            )
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<RedshiftBasics>();

        Wrapper = host.Services.GetRequiredService<RedshiftWrapper>();

        await RunScenarioAsync();
    }

    /// <summary>
    /// Run the complete Amazon Redshift scenario.
    /// </summary>
    public static async Task RunScenarioAsync()
    {
        // Set all variables to default values
        string userName = "awsuser";
        string userPassword = "AwsUser1000";
        string clusterIdentifier = "redshift-cluster-movies";
        var databaseName = "dev";
        int recordCount = 50;
        int year = 2013;
        try
        {
            Console.WriteLine(

"================================================================================");
            Console.WriteLine("Welcome to the Amazon Redshift SDK Getting Started
scenario.");
            Console.WriteLine(
                "This .NET program demonstrates how to interact with Amazon Redshift
by using the AWS SDK for .NET.");
            Console.WriteLine("Let's get started...");
            Console.WriteLine(
```

```
 "================================================================================");

            // Step 1: Get user credentials (if interactive)
            if (IsInteractive)
            {
                Console.WriteLine("Please enter a user name for the cluster (default
 is awsuser):");
                var userInput = Console.ReadLine();
                if (!string.IsNullOrEmpty(userInput))
                    userName = userInput;


 Console.WriteLine("================================================================================
                Console.WriteLine("Please enter a user password for the cluster
 (default is AwsUser1000):");
                var passwordInput = Console.ReadLine();
                if (!string.IsNullOrEmpty(passwordInput))
                    userPassword = passwordInput;


 Console.WriteLine("================================================================================

                // Step 2: Get cluster identifier
                Console.WriteLine("Enter a cluster id value (default is redshift-
cluster-movies):");
                var clusterInput = Console.ReadLine();
                if (!string.IsNullOrEmpty(clusterInput))
                    clusterIdentifier = clusterInput;
            }
            else
            {
                Console.WriteLine($"Using default values: userName={userName},
 clusterIdentifier={clusterIdentifier}");
            }

            // Step 3: Create Redshift cluster
            await Wrapper!.CreateClusterAsync(clusterIdentifier, databaseName,
 userName, userPassword);

 Console.WriteLine("================================================================================

            // Step 4: Wait for cluster to become available
```

```
Console.WriteLine("==========================================================================
            await Wrapper.WaitForClusterAvailableAsync(clusterIdentifier);

Console.WriteLine("==========================================================================

            // Step 5: List databases

Console.WriteLine("==========================================================================
            Console.WriteLine($" When you created {clusterIdentifier}, the dev
database is created by default and used in this scenario.");
            Console.WriteLine(" To create a custom database, you need to have a
CREATEDB privilege.");
            Console.WriteLine(" For more information, see the documentation here:
https://docs.aws.amazon.com/redshift/latest/dg/r_CREATE_DATABASE.html.");
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }

Console.WriteLine("==========================================================================


Console.WriteLine("==========================================================================
            Console.WriteLine($"List databases in {clusterIdentifier}");
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }
            await Wrapper.ListDatabasesAsync(clusterIdentifier, userName,
databaseName);

Console.WriteLine("==========================================================================

            // Step 6: Create Movies table

Console.WriteLine("==========================================================================
            Console.WriteLine("Now you will create a table named Movies.");
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
```

```
            }
            await Wrapper.CreateTableAsync(clusterIdentifier, databaseName,
userName);

Console.WriteLine("================================================================================

            // Step 7: Populate the Movies table

Console.WriteLine("================================================================================
            Console.WriteLine("Populate the Movies table using the Movies.json
file.");

            if (IsInteractive)
            {
                Console.WriteLine("Specify the number of records you would like to
add to the Movies Table.");
                Console.WriteLine("Please enter a value between 50 and 200.");
                Console.Write("Enter a value: ");

                var recordCountInput = Console.ReadLine();
                if (int.TryParse(recordCountInput, out var inputCount) && inputCount
is >= 50 and <= 200)
                {
                    recordCount = inputCount;
                }
                else
                {
                    Console.WriteLine($"Invalid input. Using default value of
{recordCount}.");
                }
            }
            else
            {
                Console.WriteLine($"Using default record count: {recordCount}");
            }

            await PopulateMoviesTableAsync(clusterIdentifier, databaseName,
userName, recordCount);
            Console.WriteLine($"{recordCount} records were added to the Movies
table.");

Console.WriteLine("================================================================================

            // Step 8 & 9: Query movies by year
```

```
 Console.WriteLine("=========================================================================
            Console.WriteLine("Query the Movies table by year. Enter a value between
 2012-2014.");

            if (IsInteractive)
            {
                Console.Write("Enter a year: ");
                var yearInput = Console.ReadLine();
                if (int.TryParse(yearInput, out var inputYear) && inputYear is >=
 2012 and <= 2014)
                {
                    year = inputYear;
                }
                else
                {
                    Console.WriteLine($"Invalid input. Using default value of
 {year}.");
                }
            }
            else
            {
                Console.WriteLine($"Using default year: {year}");
            }

            await Wrapper.QueryMoviesByYearAsync(clusterIdentifier, databaseName,
 userName, year);

 Console.WriteLine("=========================================================================

            // Step 10: Modify the cluster

 Console.WriteLine("=========================================================================
            Console.WriteLine("Now you will modify the Redshift cluster.");
            if (IsInteractive)
            {
                Console.WriteLine("Press Enter to continue...");
                Console.ReadLine();
            }
            await Wrapper.ModifyClusterAsync(clusterIdentifier, "wed:07:30-
wed:08:00");

 Console.WriteLine("=========================================================================
```

```
            // Step 11 & 12: Delete cluster confirmation

Console.WriteLine("=============================================================================
            if (IsInteractive)
            {
                Console.WriteLine("Would you like to delete the Amazon Redshift
cluster? (y/n)");
                var deleteResponse = Console.ReadLine();
                if (deleteResponse?.ToLower() == "y")
                {
                    await
Wrapper.DeleteClusterWithoutSnapshotAsync(clusterIdentifier);
                }
            }
            else
            {
                Console.WriteLine("Deleting the Amazon Redshift cluster...");
                await Wrapper.DeleteClusterWithoutSnapshotAsync(clusterIdentifier);
            }

Console.WriteLine("=============================================================================



Console.WriteLine("=============================================================================
            Console.WriteLine("This concludes the Amazon Redshift SDK Getting
Started scenario.");

Console.WriteLine("=============================================================================
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred during the scenario:
{ex.Message}");
            Console.WriteLine("Deleting the Amazon Redshift cluster...");
            await Wrapper!.DeleteClusterWithoutSnapshotAsync(clusterIdentifier);
            throw;
        }
    }

    /// <summary>
    /// Populate the Movies table with data from the JSON file.
    /// </summary>
    /// <param name="clusterIdentifier">The cluster identifier.</param>
    /// <param name="database">The database name.</param>
```

```
    /// <param name="dbUser">The database user.</param>
    /// <param name="recordCount">Number of records to insert.</param>
    private static async Task PopulateMoviesTableAsync(string clusterIdentifier,
 string database, string dbUser, int recordCount)
    {
        if (!File.Exists(_moviesFilePath))
        {
            throw new FileNotFoundException($"Required movies data file not found
 at: {_moviesFilePath}");
        }

        var jsonContent = await File.ReadAllTextAsync(_moviesFilePath);
        var options = new JsonSerializerOptions
        {
            PropertyNameCaseInsensitive = true
        };
        var movies = JsonSerializer.Deserialize<List<Movie>>(jsonContent, options);

        if (movies == null || movies.Count == 0)
        {
            throw new InvalidOperationException("Failed to parse movies JSON file or
 file is empty.");
        }

        var insertCount = Math.Min(recordCount, movies.Count);

        for (int i = 0; i < insertCount; i++)
        {
            var movie = movies[i];
            await Wrapper!.InsertMovieAsync(clusterIdentifier, database, dbUser, i,
 movie.Title, movie.Year);
        }
    }

    /// <summary>
    /// Movie data model.
    /// </summary>
    private class Movie
    {
        public string Title { get; set; } = string.Empty;
        public int Year { get; set; }
    }
}
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para .NET .
  - CreateCluster
  - DescribeClusters
  - DescribeStatement
  - ExecuteStatement
  - GetStatementResult
  - ListDatabasesPaginator
  - ModifyCluster

Ações

`CreateCluster`

O código de exemplo a seguir mostra como usar `CreateCluster`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Create a new Amazon Redshift cluster.
    /// </summary>
    /// <param name="clusterIdentifier">The identifier for the cluster.</param>
    /// <param name="databaseName">The name of the database.</param>
    /// <param name="masterUsername">The master username.</param>
    /// <param name="masterUserPassword">The master user password.</param>
    /// <param name="nodeType">The node type for the cluster.</param>
    /// <returns>The cluster that was created.</returns>
    public async Task<Cluster> CreateClusterAsync(string clusterIdentifier, string
databaseName,
```

```
        string masterUsername, string masterUserPassword, string nodeType =
"ra3.large")
    {
        try
        {
            var request = new CreateClusterRequest
            {
                ClusterIdentifier = clusterIdentifier,
                DBName = databaseName,
                MasterUsername = masterUsername,
                MasterUserPassword = masterUserPassword,
                NodeType = nodeType,
                NumberOfNodes = 1,
                ClusterType = "single-node"
            };

            var response = await _redshiftClient.CreateClusterAsync(request);
            Console.WriteLine($"Created cluster {clusterIdentifier}");
            return response.Cluster;
        }
        catch (ClusterAlreadyExistsException ex)
        {
            Console.WriteLine($"Cluster already exists: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't create cluster. Here's why: {ex.Message}");
            throw;
        }
    }
```

- Para obter detalhes da API, consulte CreateClustera Referência AWS SDK para .NET da API.

`DeleteCluster`

O código de exemplo a seguir mostra como usar `DeleteCluster`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
   /// <summary>
   /// Delete an Amazon Redshift cluster without a final snapshot.
   /// </summary>
   /// <param name="clusterIdentifier">The identifier for the cluster.</param>
   /// <returns>True if successful.</returns>
   public async Task<bool> DeleteClusterWithoutSnapshotAsync(string
clusterIdentifier)
   {
       try
       {
           var request = new DeleteClusterRequest
           {
               ClusterIdentifier = clusterIdentifier,
               SkipFinalClusterSnapshot = true
           };

           var response = await _redshiftClient.DeleteClusterAsync(request);
           Console.WriteLine($"The {clusterIdentifier} was deleted");
           return true;
       }
       catch (ClusterNotFoundException ex)
       {
           Console.WriteLine($"Cluster not found: {ex.Message}");
           return false;
       }
       catch (Exception ex)
       {
           Console.WriteLine($"Couldn't delete cluster. Here's why: {ex.Message}");
           return false;
       }
   }
```

- Para obter detalhes da API, consulte DeleteClustera Referência AWS SDK para .NET da API.

`DescribeClusters`

O código de exemplo a seguir mostra como usar `DescribeClusters`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Describe Amazon Redshift clusters.
    /// </summary>
    /// <param name="clusterIdentifier">Optional cluster identifier to describe a
specific cluster.</param>
    /// <returns>A list of clusters.</returns>
    public async Task<List<Cluster>> DescribeClustersAsync(string? clusterIdentifier
= null)
    {
        try
        {
            var clusters = new List<Cluster>();
            var request = new DescribeClustersRequest();
            if (!string.IsNullOrEmpty(clusterIdentifier))
            {
                request.ClusterIdentifier = clusterIdentifier;
            }

            var clustersPaginator =
_redshiftClient.Paginators.DescribeClusters(request);
            await foreach (var response in clustersPaginator.Responses)
            {
                if (response.Clusters != null)
                    clusters.AddRange(response.Clusters);
            }

            Console.WriteLine($"{clusters.Count} cluster(s) retrieved.");
            foreach (var cluster in clusters)
            {
                Console.WriteLine($"\t{cluster.ClusterIdentifier} (Status:
{cluster.ClusterStatus})");
```

```
            }

            return clusters;
        }
        catch (ClusterNotFoundException ex)
        {
            Console.WriteLine($"Cluster {clusterIdentifier} not found:
{ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't describe clusters. Here's why:
{ex.Message}");
            throw;
        }
    }
```

- Para obter detalhes da API, consulte DescribeClustersa Referência AWS SDK para .NET da API.

`DescribeStatement`

O código de exemplo a seguir mostra como usar `DescribeStatement`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Describe a statement execution.
    /// </summary>
    /// <param name="statementId">The statement ID.</param>
    /// <returns>The statement description.</returns>
    public async Task<DescribeStatementResponse> DescribeStatementAsync(string
statementId)
```

```
    {
        try
        {
            var request = new DescribeStatementRequest
            {
                Id = statementId
            };

            var response = await
_redshiftDataClient.DescribeStatementAsync(request);
            return response;
        }
        catch (Amazon.RedshiftDataAPIService.Model.ResourceNotFoundException ex)
        {
            Console.WriteLine($"Statement not found: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't describe statement. Here's why:
{ex.Message}");
            throw;
        }
    }
```

- Para obter detalhes da API, consulte DescribeStatementa Referência AWS SDK para .NET da API.

`GetStatementResult`

O código de exemplo a seguir mostra como usar `GetStatementResult`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Get the results of a statement execution.
    /// </summary>
    /// <param name="statementId">The statement ID.</param>
    /// <returns>A list of result rows.</returns>
    public async Task<List<List<Field>>> GetStatementResultAsync(string statementId)
    {
        try
        {
            var request = new GetStatementResultRequest
            {
                Id = statementId
            };

            var response = await
_redshiftDataClient.GetStatementResultAsync(request);
            return response.Records;
        }
        catch (Amazon.RedshiftDataAPIService.Model.ResourceNotFoundException ex)
        {
            Console.WriteLine($"Statement not found: {ex.Message}");
            throw;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't get statement result. Here's why:
{ex.Message}");
            throw;
        }
    }
```

- Para obter detalhes da API, consulte GetStatementResulta Referência AWS SDK para .NET da API.

`ListDatabases`

O código de exemplo a seguir mostra como usar `ListDatabases`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// List databases in a Redshift cluster.
    /// </summary>
    /// <param name="clusterIdentifier">The cluster identifier.</param>
    /// <param name="dbUser">The database user.</param>
    /// <param name="dbUser">The database name for authentication.</param>
    /// <returns>A list of database names.</returns>
    public async Task<List<string>> ListDatabasesAsync(string clusterIdentifier,
string dbUser, string databaseName)
    {
        try
        {
            var request = new ListDatabasesRequest
            {
                ClusterIdentifier = clusterIdentifier,
                DbUser = dbUser,
                Database = databaseName
            };

            var response = await _redshiftDataClient.ListDatabasesAsync(request);
            var databases = new List<string>();

            foreach (var database in response.Databases)
            {
                Console.WriteLine($"The database name is : {database}");
                databases.Add(database);
            }

            return databases;
        }
        catch (Amazon.RedshiftDataAPIService.Model.ValidationException ex)
        {
            Console.WriteLine($"Validation error: {ex.Message}");
            throw;
```

```
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't list databases. Here's why: {ex.Message}");
            throw;
        }
    }
```

- Para obter detalhes da API, consulte ListDatabasesa Referência AWS SDK para .NET da API.

`ModifyCluster`

O código de exemplo a seguir mostra como usar `ModifyCluster`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Modify an Amazon Redshift cluster.
    /// </summary>
    /// <param name="clusterIdentifier">The identifier for the cluster.</param>
    /// <param name="preferredMaintenanceWindow">The preferred maintenance window.</
param>
    /// <returns>True if successful.</returns>
    public async Task<bool> ModifyClusterAsync(string clusterIdentifier, string
 preferredMaintenanceWindow)
    {
        try
        {
            var request = new ModifyClusterRequest
            {
                ClusterIdentifier = clusterIdentifier,
                PreferredMaintenanceWindow = preferredMaintenanceWindow
            };

            var response = await _redshiftClient.ModifyClusterAsync(request);
```

```
            Console.WriteLine($"The modified cluster was successfully modified and
has {response.Cluster.PreferredMaintenanceWindow} as the maintenance window");
            return true;
        }
        catch (ClusterNotFoundException ex)
        {
            Console.WriteLine($"Cluster {clusterIdentifier} not found:
{ex.Message}");
            return false;
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't modify cluster. Here's why: {ex.Message}");
            return false;
        }
    }
```

- Para obter detalhes da API, consulte ModifyClustera Referência AWS SDK para .NET da API.

Exemplos do Amazon S3 usando SDK para .NET (v4)

Os exemplos de código a seguir mostram como realizar ações e implementar cenários comuns usando o AWS SDK para .NET (v4) com o Amazon S3.

As noções básicas são exemplos de código que mostram como realizar as operações essenciais em um serviço.

Ações são trechos de código de programas maiores e devem ser executadas em contexto. Embora as ações mostrem como chamar perfis de serviço individuais, você pode ver as ações no contexto em seus cenários relacionados.

Cenários são exemplos de código que mostram como realizar tarefas específicas chamando várias funções dentro de um serviço ou combinadas com outros Serviços da AWS.

Cada exemplo inclui um link para o código-fonte completo, em que você pode encontrar instruções sobre como configurar e executar o código.

Tópicos

- Conceitos básicos

- Conceitos básicos
- Ações
- Cenários

Conceitos básicos

Olá, Amazon S3

O exemplo de código a seguir mostra como começar a usar o Amazon S3.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
/// <summary>
/// Hello Amazon Simple Storage Service
// (Amazon S3) example.
/// </summary>
public class HelloS3
{
    /// <summary>
    /// Main method to run the Hello S3 example.
    /// </summary>
    /// <param name="args">Command line arguments.</param>
    /// <returns>A Task object.</returns>
    public static async Task Main(string[] args)
    {
        var s3Client = new AmazonS3Client();

        try
        {
            Console.WriteLine("Hello Amazon S3! Let's list your buckets:");
            Console.WriteLine(new string('-', 80));

            // Use the built-in paginator to list buckets
            var request = new ListBucketsRequest();
```

```
            var paginator = s3Client.Paginators.ListBuckets(request);

            var buckets = new List<S3Bucket>();

            await foreach (var response in paginator.Responses)
            {
                buckets.AddRange(response.Buckets);
            }

            if (buckets.Any())
            {
                Console.WriteLine($"Found {buckets.Count} S3 buckets:");
                Console.WriteLine();

                foreach (var bucket in buckets)
                {
                    Console.WriteLine($"- Bucket Name: {bucket.BucketName}");
                    Console.WriteLine($"  Creation Date: {bucket.CreationDate:yyyy-
MM-dd HH:mm:ss UTC}");
                    Console.WriteLine();
                }
            }
            else
            {
                Console.WriteLine("No S3 buckets found in your account.");
            }

            Console.WriteLine("Hello S3 completed successfully.");
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"S3 service error occurred: {ex.Message}");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Couldn't list S3 buckets. Here's why:
 {ex.Message}");
        }
    }
}
```

- Para obter detalhes da API, consulte ListBucketsa Referência AWS SDK para .NET da API.

Conceitos básicos

Conheça os conceitos básicos

O exemplo de código a seguir mostra como:

- Criar um bucket e fazer upload de um arquivo para ele.
- Baixar um objeto de um bucket.
- Copiar um objeto em uma subpasta em um bucket.
- Listar os objetos em um bucket.
- Exclua os objetos do bucket e o bucket.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Execute um cenário interativo que demonstre os recursos do Amazon S3.

```
public class S3_Basics
{
    public static bool IsInteractive = true;
    public static string BucketName = null!;
    public static string TempFilePath = null!;
    public static S3Wrapper _s3Wrapper = null!;
    public static ILogger<S3_Basics> _logger = null!;

    public static async Task Main(string[] args)
    {
        // Set up dependency injection for the Amazon service.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonS3>()
                    .AddTransient<S3Wrapper>()
                    .AddLogging(builder => builder.AddConsole()))
            .Build();
```

```
        _logger = LoggerFactory.Create(builder => builder.AddConsole())
            .CreateLogger<S3_Basics>();

        _s3Wrapper = host.Services.GetRequiredService<S3Wrapper>();

        var sepBar = new string('-', 45);

        Console.WriteLine(sepBar);
        Console.WriteLine("Amazon Simple Storage Service (Amazon S3) basic");
        Console.WriteLine("procedures. This application will:");
        Console.WriteLine("\n\t1. Create a bucket");
        Console.WriteLine("\n\t2. Upload an object to the new bucket");
        Console.WriteLine("\n\t3. Copy the uploaded object to a folder in the
bucket");
        Console.WriteLine("\n\t4. List the items in the new bucket");
        Console.WriteLine("\n\t5. Delete all the items in the bucket");
        Console.WriteLine("\n\t6. Delete the bucket");
        Console.WriteLine(sepBar);

        await RunScenario(_s3Wrapper, _logger);

        Console.WriteLine(sepBar);
        Console.WriteLine("The Amazon S3 scenario has successfully completed.");
        Console.WriteLine(sepBar);
    }

    /// <summary>
    /// Run the S3 Basics scenario with injected dependencies.
    /// </summary>
    /// <param name="s3Wrapper">The S3 wrapper instance.</param>
    /// <param name="scenarioLogger">The logger instance.</param>
    /// <returns>A Task object.</returns>
    public static async Task RunScenario(S3Wrapper s3Wrapper, ILogger<S3_Basics>
scenarioLogger)
    {
        string bucketName = BucketName;
        string filePath = TempFilePath;
        string keyName = string.Empty;

        var sepBar = new string('-', 45);

        try
        {
            // Create a bucket.
```

```
            Console.WriteLine($"\n{sepBar}");
            Console.WriteLine("\nCreate a new Amazon S3 bucket.\n");
            Console.WriteLine(sepBar);

            if (IsInteractive)
            {
                Console.Write("Please enter a name for the new bucket: ");
                bucketName = Console.ReadLine();
            }
            else
            {
                Console.WriteLine($"Using bucket name: {bucketName}");
            }

            var success = await s3Wrapper.CreateBucketAsync(bucketName);
            if (success)
            {
                Console.WriteLine($"Successfully created bucket: {bucketName}.\n");
            }
            else
            {
                Console.WriteLine($"Could not create bucket: {bucketName}.\n");
            }

            Console.WriteLine(sepBar);
            Console.WriteLine("Upload a file to the new bucket.");
            Console.WriteLine(sepBar);

            if (IsInteractive)
            {
                // Get the local path and filename for the file to upload.
                while (string.IsNullOrEmpty(filePath))
                {
                    Console.Write("Please enter the path and filename of the file to
 upload: ");
                    filePath = Console.ReadLine();

                    // Confirm that the file exists on the local computer.
                    if (!File.Exists(filePath))
                    {
                        Console.WriteLine($"Couldn't find {filePath}. Try again.
\n");
                        filePath = string.Empty;
                    }
```

```
                }
            }
            else
            {
                // Use the public variable if set, otherwise create a temp file
                if (!string.IsNullOrEmpty(TempFilePath))
                {
                    filePath = TempFilePath;
                    Console.WriteLine($"Using provided test file: {filePath}");
                }
                else
                {
                    // Create a temporary test file for non-interactive mode
                    filePath = Path.GetTempFileName();
                    var testContent = "This is a test file for S3 basics scenario.
\nGenerated on: " + DateTime.UtcNow.ToString("yyyy-MM-dd HH:mm:ss UTC");
                    await File.WriteAllTextAsync(filePath, testContent);
                    Console.WriteLine($"Created temporary test file: {filePath}");
                }
            }

            // Get the file name from the full path.
            keyName = Path.GetFileName(filePath);

            success = await s3Wrapper.UploadFileAsync(bucketName, keyName,
 filePath);

            if (success)
            {
                Console.WriteLine($"Successfully uploaded {keyName} from {filePath}
 to {bucketName}.\n");
            }
            else
            {
                Console.WriteLine($"Could not upload {keyName}.\n");
            }

            // Set up download path
            string downloadPath = string.Empty;

            if (IsInteractive)
            {
                // Now get a new location where we can save the file.
                while (string.IsNullOrEmpty(downloadPath))
```

```
                {
                    // First get the path to which the file will be downloaded.
                    Console.Write("Please enter the path where the file will be
downloaded: ");
                    downloadPath = Console.ReadLine();

                    // Confirm that the file doesn't already exist on the local
computer.
                    if (File.Exists($"{downloadPath}\\{keyName}"))
                    {
                        Console.WriteLine($"Sorry, the file already exists in that
location.\n");
                        downloadPath = string.Empty;
                    }
                }
            }
            else
            {
                downloadPath = Path.GetTempPath();
                var downloadFile = Path.Combine(downloadPath, keyName);
                if (File.Exists(downloadFile))
                {
                    File.Delete(downloadFile);
                }

                Console.WriteLine($"Using download path: {downloadPath}");
            }

            // Download an object from a bucket.
            success = await s3Wrapper.DownloadObjectFromBucketAsync(bucketName,
keyName, downloadPath);

            if (success)
            {
                Console.WriteLine($"Successfully downloaded {keyName}.\n");
            }
            else
            {
                Console.WriteLine($"Sorry, could not download {keyName}.\n");
            }

            // Copy the object to a different folder in the bucket.
            string folderName = string.Empty;
```

```
            if (IsInteractive)
            {
                while (string.IsNullOrEmpty(folderName))
                {
                    Console.Write("Please enter the name of the folder to copy your
 object to: ");
                    folderName = Console.ReadLine();
                }
            }
            else
            {
                folderName = "test-folder";
                Console.WriteLine($"Using folder name: {folderName}");
            }

            await s3Wrapper.CopyObjectInBucketAsync(bucketName, keyName,
 folderName);

            // List the objects in the bucket.
            await s3Wrapper.ListBucketContentsAsync(bucketName);

            // Delete the contents of the bucket.
            if (IsInteractive)
            {
                Console.WriteLine("Press <Enter> when you are ready to delete the
 bucket contents.");
                _ = Console.ReadLine();
            }

            var deleteContentsSuccess = await
 s3Wrapper.DeleteBucketContentsAsync(bucketName);
            if (deleteContentsSuccess)
            {
                Console.WriteLine($"Successfully deleted contents of {bucketName}.
\n");
            }
            else
            {
                Console.WriteLine($"Sorry, could not delete contents of
 {bucketName}.\n");
            }

            if (IsInteractive)
            {
```

```
                // Deleting the bucket too quickly after separately deleting its
contents can
                // cause an error that the bucket isn't empty. To delete contents
and bucket in one
                // operation, use AmazonS3Util.DeleteS3BucketWithObjectsAsync
                Console.WriteLine("Press <Enter> when you are ready to delete the
bucket.");
                _ = Console.ReadLine();
            }
            else
            {
                // Add a small delay for non-interactive mode to ensure objects are
fully deleted.
                Console.WriteLine("Waiting a moment for objects to be fully
deleted...");
                await Task.Delay(2000);
            }

            // Delete the bucket.
            var deleteSuccess = await s3Wrapper.DeleteBucketAsync(bucketName);
            if (deleteSuccess)
            {
                Console.WriteLine($"Successfully deleted {bucketName}.\n");
            }
            else
            {
                Console.WriteLine($"Sorry, could not delete {bucketName}.\n");
            }

            // Clean up temporary files in non-interactive mode
            if (!IsInteractive)
            {
                try
                {
                    if (File.Exists(filePath))
                    {
                        File.Delete(filePath);
                        Console.WriteLine("Cleaned up temporary test file.");
                    }

                    var downloadFile = Path.Combine(downloadPath, keyName);
                    if (File.Exists(downloadFile))
                    {
                        File.Delete(downloadFile);
```

```
                    Console.WriteLine("Cleaned up downloaded test file.");
                }
            }
            catch (Exception ex)
            {
                scenarioLogger.LogWarning(ex, "Failed to clean up temporary
files.");
            }
        }
    }
    catch (Exception ex)
    {
        scenarioLogger.LogError(ex, "An error occurred during the S3 scenario
execution.");

        // Clean up on error - delete bucket if it exists
        try
        {
            if (!string.IsNullOrEmpty(bucketName))
            {
                await s3Wrapper.DeleteBucketContentsAsync(bucketName);
                await s3Wrapper.DeleteBucketAsync(bucketName);
            }
        }
        catch (Exception cleanupEx)
        {
            scenarioLogger.LogError(cleanupEx, "Error during cleanup.");
        }

        // Clean up temporary files in non-interactive mode
        if (!IsInteractive)
        {
            try
            {
                if (!string.IsNullOrEmpty(filePath) && File.Exists(filePath))
                {
                    File.Delete(filePath);
                }
            }
            catch (Exception fileCleanupEx)
            {
                scenarioLogger.LogWarning(fileCleanupEx, "Failed to clean up
temporary files during error handling.");
            }
```

```
            }

            throw;
        }
    }
}
```

Uma classe de wrapper para os métodos de SDK do Amazon S3.

```
using Amazon.S3;
using Amazon.S3.Model;

namespace S3_Actions;

/// <summary>
/// This class contains all of the methods for working with Amazon Simple
/// Storage Service (Amazon S3) buckets.
/// </summary>
public class S3Wrapper
{
    private readonly IAmazonS3 _amazonS3;

    /// <summary>
    /// Initializes a new instance of the <see cref="S3Wrapper"/> class.
    /// </summary>
    /// <param name="amazonS3">An initialized Amazon S3 client object.</param>
    public S3Wrapper(IAmazonS3 amazonS3)
    {
        _amazonS3 = amazonS3;
    }


    /// <summary>
    /// Shows how to create a new Amazon S3 bucket.
    /// </summary>
    /// <param name="bucketName">The name of the bucket to create.</param>
    /// <returns>A boolean value representing the success or failure of
    /// the bucket creation process.</returns>
    public async Task<bool> CreateBucketAsync(string bucketName)
    {
        try
```

```
        {
            var request = new PutBucketRequest
            {
                BucketName = bucketName,
                UseClientRegion = true,
            };

            var response = await _amazonS3.PutBucketAsync(request);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error creating bucket: '{ex.Message}'");
            return false;
        }
    }



    /// <summary>
    /// Shows how to upload a file from the local computer to an Amazon S3
    /// bucket.
    /// </summary>
    /// <param name="bucketName">The Amazon S3 bucket to which the object
    /// will be uploaded.</param>
    /// <param name="objectName">The object to upload.</param>
    /// <param name="filePath">The path, including file name, of the object
    /// on the local computer to upload.</param>
    /// <returns>A boolean value indicating the success or failure of the
    /// upload procedure.</returns>
    public async Task<bool> UploadFileAsync(
        string bucketName,
        string objectName,
        string filePath)
    {
        try
        {
            var request = new PutObjectRequest
            {
                BucketName = bucketName,
                Key = objectName,
                FilePath = filePath,
            };
```

```
            var response = await _amazonS3.PutObjectAsync(request);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error uploading {objectName}: {ex.Message}");
            return false;
        }
    }



    /// <summary>
    /// Shows how to download an object from an Amazon S3 bucket to the
    /// local computer.
    /// </summary>
    /// <param name="bucketName">The name of the bucket where the object is
    /// currently stored.</param>
    /// <param name="objectName">The name of the object to download.</param>
    /// <param name="filePath">The path, including filename, where the
    /// downloaded object will be stored.</param>
    /// <returns>A boolean value indicating the success or failure of the
    /// download process.</returns>
    public async Task<bool> DownloadObjectFromBucketAsync(
        string bucketName,
        string objectName,
        string filePath)
    {
        var request = new GetObjectRequest
        {
            BucketName = bucketName,
            Key = objectName,
        };

        using GetObjectResponse response = await _amazonS3.GetObjectAsync(request);

        try
        {
            // Save object to local file
            await response.WriteResponseStreamToFileAsync($"{filePath}\
\{objectName}", true, CancellationToken.None);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
```

```
        {
            Console.WriteLine($"Error saving {objectName}: {ex.Message}");
            return false;
        }
    }



    /// <summary>
    /// Copies an object in an Amazon S3 bucket to a folder within the
    /// same bucket.
    /// </summary>
    /// <param name="bucketName">The name of the Amazon S3 bucket where the
    /// object to copy is located.</param>
    /// <param name="objectName">The object to be copied.</param>
    /// <param name="folderName">The folder to which the object will
    /// be copied.</param>
    /// <returns>A boolean value that indicates the success or failure of
    /// the copy operation.</returns>
    public async Task<bool> CopyObjectInBucketAsync(
        string bucketName,
        string objectName,
        string folderName)
    {
        try
        {
            var request = new CopyObjectRequest
            {
                SourceBucket = bucketName,
                SourceKey = objectName,
                DestinationBucket = bucketName,
                DestinationKey = $"{folderName}\\{objectName}",
            };
            var response = await _amazonS3.CopyObjectAsync(request);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error copying object: '{ex.Message}'");
            return false;
        }
    }
```

```
   /// <summary>
   /// Shows how to list the objects in an Amazon S3 bucket.
   /// </summary>
   /// <param name="bucketName">The name of the bucket for which to list.
   /// <param name="printList">True to print out the list.
   /// <returns>The collection of objects.</returns>
   public async Task<List<S3Object>?> ListBucketContentsAsync(string bucketName,
bool printList = true)
   {
       try
       {
           var request = new ListObjectsV2Request
           {
               BucketName = bucketName,
               MaxKeys = 5,
           };

           if (printList)
           {
               Console.WriteLine("--------------------------------------");
               Console.WriteLine($"Listing the contents of {bucketName}:");
               Console.WriteLine("--------------------------------------");
           }

           var listObjectsV2Paginator = _amazonS3.Paginators.ListObjectsV2(new
ListObjectsV2Request
           {
               BucketName = bucketName,
           });
           var s3Objects = new List<S3Object>();
           await foreach (var response in listObjectsV2Paginator.Responses)
           {
               if (response.S3Objects != null)
               {
                   s3Objects.AddRange(response.S3Objects);
               }
           }

           if (printList)
           {
               Console.WriteLine($"Number of Objects: {s3Objects.Count}");
               foreach (var entry in s3Objects)
               {
```

```
                    Console.WriteLine($"Key = {entry.Key} Size = {entry.Size}");
                }
            }

            return s3Objects;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error encountered on server. Message:'{ex.Message}'
getting list of objects.");
            return null;
        }
    }



    /// <summary>
    /// Delete all of the objects stored in an existing Amazon S3 bucket.
    /// </summary>
    /// <param name="bucketName">The name of the bucket from which the
    /// contents will be deleted.</param>
    /// <returns>A boolean value that represents the success or failure of
    /// deleting all of the objects in the bucket.</returns>
    public async Task<bool> DeleteBucketContentsAsync(string bucketName)
    {
        // Iterate over the contents of the bucket and delete all objects.
        try
        {
            // Delete all objects in the bucket.
            var deleteList = await ListBucketContentsAsync(bucketName, false);
            if (deleteList != null && deleteList.Any())
            {
                await _amazonS3.DeleteObjectsAsync(new DeleteObjectsRequest()
                {
                    BucketName = bucketName,
                    Objects = deleteList.Select(o => new KeyVersion { Key =
o.Key }).ToList(),
                });
            }

            return true;
        }
        catch (AmazonS3Exception ex)
        {
```

```
            Console.WriteLine($"Error deleting objects: {ex.Message}");
            return false;
        }
    }



    /// <summary>
    /// Shows how to delete an Amazon S3 bucket.
    /// </summary>
    /// <param name="bucketName">The name of the Amazon S3 bucket to delete.</param>
    /// <returns>A boolean value that represents the success or failure of
    /// the delete operation.</returns>
    public async Task<bool> DeleteBucketAsync(string bucketName)
    {
        try
        {
            var request = new DeleteBucketRequest { BucketName = bucketName, };

            await _amazonS3.DeleteBucketAsync(request);
            return true;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error deleting bucket: {ex.Message}");
            return false;
        }
    }

}
```

- Para obter detalhes da API, consulte os tópicos a seguir na Referência da API AWS SDK para .NET .
  - CopyObject
  - CreateBucket
  - DeleteBucket
  - DeleteObjects
  - GetObject
  - ListObjectsV2

- PutObject

Ações

`CopyObject`

O código de exemplo a seguir mostra como usar `CopyObject`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
/// <summary>
/// Copies an object in an Amazon S3 bucket to a folder within the
/// same bucket.
/// </summary>
/// <param name="bucketName">The name of the Amazon S3 bucket where the
/// object to copy is located.</param>
/// <param name="objectName">The object to be copied.</param>
/// <param name="folderName">The folder to which the object will
/// be copied.</param>
/// <returns>A boolean value that indicates the success or failure of
/// the copy operation.</returns>
public async Task<bool> CopyObjectInBucketAsync(
    string bucketName,
    string objectName,
    string folderName)
{
    try
    {
        var request = new CopyObjectRequest
        {
            SourceBucket = bucketName,
            SourceKey = objectName,
            DestinationBucket = bucketName,
            DestinationKey = $"{folderName}\\{objectName}",
```

```
            };
            var response = await _amazonS3.CopyObjectAsync(request);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error copying object: '{ex.Message}'");
            return false;
        }
    }
```

- Para obter detalhes da API, consulte CopyObjecta Referência AWS SDK para .NET da API.

`CreateBucket`

O código de exemplo a seguir mostra como usar `CreateBucket`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
/// <summary>
/// Shows how to create a new Amazon S3 bucket.
/// </summary>
/// <param name="bucketName">The name of the bucket to create.</param>
/// <returns>A boolean value representing the success or failure of
/// the bucket creation process.</returns>
public async Task<bool> CreateBucketAsync(string bucketName)
{
    try
    {
        var request = new PutBucketRequest
        {
            BucketName = bucketName,
            UseClientRegion = true,
```

```
            };

            var response = await _amazonS3.PutBucketAsync(request);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error creating bucket: '{ex.Message}'");
            return false;
        }
    }
```

- Para obter detalhes da API, consulte CreateBucketa Referência AWS SDK para .NET da API.

`CreatePresignedPost`

O código de exemplo a seguir mostra como usar `CreatePresignedPost`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Crie um URL POST pré-assinado.

```
    /// <summary>
    /// Create a presigned POST URL with conditions.
    /// </summary>
    /// <param name="s3Client">The Amazon S3 client.</param>
    /// <param name="bucketName">The name of the bucket.</param>
    /// <param name="objectKey">The object key (path) where the uploaded file will
 be stored.</param>
    /// <param name="expires">When the presigned URL expires.</param>
    /// <param name="fields">Dictionary of fields to add to the form.</param>
    /// <param name="conditions">List of conditions to apply.</param>
    /// <returns>A CreatePresignedPostResponse object with URL and form fields.</
returns>
```

```
    public async Task<CreatePresignedPostResponse> CreatePresignedPostAsync(
        IAmazonS3 s3Client,
        string bucketName,
        string objectKey,
        DateTime expires,
        Dictionary<string, string>? fields = null,
        List<S3PostCondition>? conditions = null)
    {
        var request = new CreatePresignedPostRequest
        {
            BucketName = bucketName,
            Key = objectKey,
            Expires = expires
        };

        // Add custom fields if provided
        if (fields != null)
        {
            foreach (var field in fields)
            {
                request.Fields.Add(field.Key, field.Value);
            }
        }

        // Add conditions if provided
        if (conditions != null)
        {
            foreach (var condition in conditions)
            {
                request.Conditions.Add(condition);
            }
        }

        return await s3Client.CreatePresignedPostAsync(request);
    }
```

- Para obter detalhes da API, consulte CreatePresignedPosta Referência AWS SDK para .NET da API.

`DeleteBucket`

O código de exemplo a seguir mostra como usar `DeleteBucket`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
/// <summary>
/// Shows how to delete an Amazon S3 bucket.
/// </summary>
/// <param name="bucketName">The name of the Amazon S3 bucket to delete.</param>
/// <returns>A boolean value that represents the success or failure of
/// the delete operation.</returns>
public async Task<bool> DeleteBucketAsync(string bucketName)
{
    try
    {
        var request = new DeleteBucketRequest { BucketName = bucketName, };

        await _amazonS3.DeleteBucketAsync(request);
        return true;
    }
    catch (AmazonS3Exception ex)
    {
        Console.WriteLine($"Error deleting bucket: {ex.Message}");
        return false;
    }
}
```

- Para obter detalhes da API, consulte DeleteBucketa Referência AWS SDK para .NET da API.

`DeleteObjects`

O código de exemplo a seguir mostra como usar `DeleteObjects`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Delete all of the objects stored in an existing Amazon S3 bucket.
    /// </summary>
    /// <param name="bucketName">The name of the bucket from which the
    /// contents will be deleted.</param>
    /// <returns>A boolean value that represents the success or failure of
    /// deleting all of the objects in the bucket.</returns>
    public async Task<bool> DeleteBucketContentsAsync(string bucketName)
    {
        // Iterate over the contents of the bucket and delete all objects.
        try
        {
            // Delete all objects in the bucket.
            var deleteList = await ListBucketContentsAsync(bucketName, false);
            if (deleteList != null && deleteList.Any())
            {
                await _amazonS3.DeleteObjectsAsync(new DeleteObjectsRequest()
                {
                    BucketName = bucketName,
                    Objects = deleteList.Select(o => new KeyVersion { Key =
o.Key }).ToList(),
                });
            }

            return true;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error deleting objects: {ex.Message}");
            return false;
        }
    }
```

- Para obter detalhes da API, consulte DeleteObjectsa Referência AWS SDK para .NET da API.

`GetObject`

O código de exemplo a seguir mostra como usar `GetObject`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
/// <summary>
/// Shows how to download an object from an Amazon S3 bucket to the
/// local computer.
/// </summary>
/// <param name="bucketName">The name of the bucket where the object is
/// currently stored.</param>
/// <param name="objectName">The name of the object to download.</param>
/// <param name="filePath">The path, including filename, where the
/// downloaded object will be stored.</param>
/// <returns>A boolean value indicating the success or failure of the
/// download process.</returns>
public async Task<bool> DownloadObjectFromBucketAsync(
    string bucketName,
    string objectName,
    string filePath)
{
    var request = new GetObjectRequest
    {
        BucketName = bucketName,
        Key = objectName,
    };

    using GetObjectResponse response = await _amazonS3.GetObjectAsync(request);

    try
```

```
        {
            // Save object to local file
            await response.WriteResponseStreamToFileAsync($"{filePath}\
\{objectName}", true, CancellationToken.None);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error saving {objectName}: {ex.Message}");
            return false;
        }
    }
```

- Para obter detalhes da API, consulte GetObjecta Referência AWS SDK para .NET da API.

`ListObjectsV2`

O código de exemplo a seguir mostra como usar `ListObjectsV2`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
    /// <summary>
    /// Shows how to list the objects in an Amazon S3 bucket.
    /// </summary>
    /// <param name="bucketName">The name of the bucket for which to list.
    /// <param name="printList">True to print out the list.
    /// <returns>The collection of objects.</returns>
    public async Task<List<S3Object>?> ListBucketContentsAsync(string bucketName,
 bool printList = true)
    {
        try
        {
            var request = new ListObjectsV2Request
```

```
            {
                BucketName = bucketName,
                MaxKeys = 5,
            };

            if (printList)
            {
                Console.WriteLine("--------------------------------------");
                Console.WriteLine($"Listing the contents of {bucketName}:");
                Console.WriteLine("--------------------------------------");
            }

            var listObjectsV2Paginator = _amazonS3.Paginators.ListObjectsV2(new
ListObjectsV2Request
            {
                BucketName = bucketName,
            });
            var s3Objects = new List<S3Object>();
            await foreach (var response in listObjectsV2Paginator.Responses)
            {
                if (response.S3Objects != null)
                {
                    s3Objects.AddRange(response.S3Objects);
                }
            }

            if (printList)
            {
                Console.WriteLine($"Number of Objects: {s3Objects.Count}");
                foreach (var entry in s3Objects)
                {
                    Console.WriteLine($"Key = {entry.Key} Size = {entry.Size}");
                }
            }

            return s3Objects;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error encountered on server. Message:'{ex.Message}'
getting list of objects.");
            return null;
        }
    }
```

- Para obter detalhes da API, consulte ListObjectsV2 na Referência AWS SDK para .NET da API.

`PutObject`

O código de exemplo a seguir mostra como usar `PutObject`.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

```
/// <summary>
/// Shows how to upload a file from the local computer to an Amazon S3
/// bucket.
/// </summary>
/// <param name="bucketName">The Amazon S3 bucket to which the object
/// will be uploaded.</param>
/// <param name="objectName">The object to upload.</param>
/// <param name="filePath">The path, including file name, of the object
/// on the local computer to upload.</param>
/// <returns>A boolean value indicating the success or failure of the
/// upload procedure.</returns>
public async Task<bool> UploadFileAsync(
    string bucketName,
    string objectName,
    string filePath)
{
    try
    {
        var request = new PutObjectRequest
        {
            BucketName = bucketName,
            Key = objectName,
            FilePath = filePath,
        };
```

```
            var response = await _amazonS3.PutObjectAsync(request);
            return response.HttpStatusCode == System.Net.HttpStatusCode.OK;
        }
        catch (AmazonS3Exception ex)
        {
            Console.WriteLine($"Error uploading {objectName}: {ex.Message}");
            return false;
        }
    }
```

- Para obter detalhes da API, consulte PutObjecta Referência AWS SDK para .NET da API.

Cenários

Criar um URL pré-assinado

O exemplo de código a seguir mostra como criar um URL pré-assinado para o Amazon S3 e fazer upload de um objeto.

SDK para .NET (v4)

> **Note**
>
> Tem mais sobre GitHub. Encontre o exemplo completo e saiba como configurar e executar no AWS Code Examples Repository.

Crie e use POST pré-assinado URLs para carregamentos diretos do navegador.

```
/// <summary>
/// Scenario demonstrating the complete workflow for presigned POST URLs:
/// 1. Create an S3 bucket
/// 2. Create a presigned POST URL
/// 3. Upload a file using the presigned POST URL
/// 4. Clean up resources
/// </summary>
public class CreatePresignedPostBasics
{
    public static ILogger<CreatePresignedPostBasics> _logger = null!;
```

```
    public static S3Wrapper _s3Wrapper = null!;
    public static UiMethods _uiMethods = null!;
    public static IHttpClientFactory _httpClientFactory = null!;
    public static bool _isInteractive = true;
    public static string? _bucketName;
    public static string? _objectKey;

    /// <summary>
    /// Set up the services and logging.
    /// </summary>
    /// <param name="host">The IHost instance.</param>
    public static void SetUpServices(IHost host)
    {
        var loggerFactory = LoggerFactory.Create(builder =>
        {
            builder.AddConsole();
        });
        _logger = new Logger<CreatePresignedPostBasics>(loggerFactory);

        _s3Wrapper = host.Services.GetRequiredService<S3Wrapper>();
        _httpClientFactory = host.Services.GetRequiredService<IHttpClientFactory>();
        _uiMethods = new UiMethods();
    }

    /// <summary>
    /// Perform the actions defined for the Amazon S3 Presigned POST scenario.
    /// </summary>
    /// <param name="args">Command line arguments.</param>
    /// <returns>A Task object.</returns>
    public static async Task Main(string[] args)
    {
        _isInteractive = !args.Contains("--non-interactive");

        // Set up dependency injection for Amazon S3
        using var host =
Microsoft.Extensions.Hosting.Host.CreateDefaultBuilder(args)
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonS3>()
                    .AddTransient<S3Wrapper>()
                    .AddHttpClient()
            )
            .Build();

        SetUpServices(host);
```

```
        try
        {
            // Display overview
            _uiMethods.DisplayOverview();
            _uiMethods.PressEnter(_isInteractive);

            // Step 1: Create bucket
            await CreateBucketAsync();
            _uiMethods.PressEnter(_isInteractive);

            // Step 2: Create presigned URL
            _uiMethods.DisplayTitle("Step 2: Create presigned POST URL");
            var response = await CreatePresignedPostAsync();
            _uiMethods.PressEnter(_isInteractive);

            // Step 3: Display URL and fields
            _uiMethods.DisplayTitle("Step 3: Presigned POST URL details");
            DisplayPresignedPostFields(response);
            _uiMethods.PressEnter(_isInteractive);

            // Step 4: Upload file
            _uiMethods.DisplayTitle("Step 4: Upload test file using presigned POST
URL");
            await UploadFileAsync(response);
            _uiMethods.PressEnter(_isInteractive);

            // Step 5: Verify file exists
            await VerifyFileExistsAsync();
            _uiMethods.PressEnter(_isInteractive);

            // Step 6: Cleanup
            _uiMethods.DisplayTitle("Step 6: Clean up resources");
            await CleanupAsync();

            _uiMethods.DisplayTitle("S3 Presigned POST Scenario completed
successfully!");
            _uiMethods.PressEnter(_isInteractive);
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "Error in scenario");
            Console.WriteLine($"Error: {ex.Message}");
```

```
            // Attempt cleanup if there was an error
            if (!string.IsNullOrEmpty(_bucketName))
            {
                _uiMethods.DisplayTitle("Cleaning up resources after error");
                await _s3Wrapper.DeleteBucketAsync(_bucketName);
                Console.WriteLine($"Cleaned up bucket: {_bucketName}");
            }
        }
    }

    /// <summary>
    /// Create an S3 bucket for the scenario.
    /// </summary>
    private static async Task CreateBucketAsync()
    {
        _uiMethods.DisplayTitle("Step 1: Create an S3 bucket");

        // Generate a default bucket name for the scenario
        var defaultBucketName = $"presigned-post-demo-
{DateTime.Now:yyyyMMddHHmmss}".ToLower();

        // Prompt user for bucket name or use default in non-interactive mode
        _bucketName = _uiMethods.GetUserInput(
            $"Enter S3 bucket name (or press Enter for '{defaultBucketName}'): ",
            defaultBucketName,
            _isInteractive);

        // Basic validation to ensure bucket name is not empty
        if (string.IsNullOrWhiteSpace(_bucketName))
        {
            _bucketName = defaultBucketName;
        }

        Console.WriteLine($"Creating bucket: {_bucketName}");

        await _s3Wrapper.CreateBucketAsync(_bucketName);

        Console.WriteLine($"Successfully created bucket: {_bucketName}");
    }


    /// <summary>
    /// Create a presigned POST URL.
    /// </summary>
```

```
    private static async Task<CreatePresignedPostResponse>
 CreatePresignedPostAsync()
    {
        _objectKey = "example-upload.txt";
        var expiration = DateTime.UtcNow.AddMinutes(10); // Short expiration for the
 demo

        Console.WriteLine($"Creating presigned POST URL for {_bucketName}/
{_objectKey}");
        Console.WriteLine($"Expiration: {expiration} UTC");

        var s3Client = _s3Wrapper.GetS3Client();

        var response = await _s3Wrapper.CreatePresignedPostAsync(
            s3Client, _bucketName!, _objectKey, expiration);

        Console.WriteLine("Successfully created presigned POST URL");
        return response;
    }

    /// <summary>
    /// Upload a file using the presigned POST URL.
    /// </summary>
    private static async Task UploadFileAsync(CreatePresignedPostResponse response)
    {

        // Create a temporary test file to upload
        string testFilePath = Path.GetTempFileName();
        string testContent = "This is a test file for the S3 presigned POST
 scenario.";

        await File.WriteAllTextAsync(testFilePath, testContent);
        Console.WriteLine($"Created test file at: {testFilePath}");

        // Upload the file using the presigned POST URL
        Console.WriteLine("\nUploading file using the presigned POST URL...");
        var uploadResult = await UploadFileWithPresignedPostAsync(response,
 testFilePath);

        // Display the upload result
        if (uploadResult.Success)
        {
            Console.WriteLine($"Upload successful! Status code:
 {uploadResult.StatusCode}");
```

```
        }
        else
        {
            Console.WriteLine($"Upload failed with status code:
 {uploadResult.StatusCode}");
            Console.WriteLine($"Error: {uploadResult.Response}");
            throw new Exception("File upload failed");
        }

        // Clean up the temporary file
        File.Delete(testFilePath);
        Console.WriteLine("Temporary file deleted");
    }

    /// <summary>
    /// Helper method to upload a file using a presigned POST URL.
    /// </summary>
    private static async Task<(bool Success, HttpStatusCode StatusCode, string
 Response)> UploadFileWithPresignedPostAsync(
        CreatePresignedPostResponse response,
        string filePath)
    {
        try
        {
            _logger.LogInformation("Uploading file {filePath} using presigned POST
 URL", filePath);

            using var httpClient = _httpClientFactory.CreateClient();
            using var formContent = new MultipartFormDataContent();

            // Add all the fields from the presigned POST response
            foreach (var field in response.Fields)
            {
                formContent.Add(new StringContent(field.Value), field.Key);
            }

            // Add the file content
            var fileStream = File.OpenRead(filePath);
            var fileName = Path.GetFileName(filePath);
            var fileContent = new StreamContent(fileStream);
            fileContent.Headers.ContentType = new MediaTypeHeaderValue("text/
plain");
            formContent.Add(fileContent, "file", fileName);
```

```
            // Send the POST request
            var httpResponse = await httpClient.PostAsync(response.Url,
 formContent);
            var responseContent = await httpResponse.Content.ReadAsStringAsync();

            // Log and return the result
            _logger.LogInformation("Upload completed with status code {statusCode}",
 httpResponse.StatusCode);

            return (httpResponse.IsSuccessStatusCode, httpResponse.StatusCode,
 responseContent);
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "Error uploading file");
            return (false, HttpStatusCode.InternalServerError, ex.Message);
        }
    }

    /// <summary>
    /// Verify that the uploaded file exists in the S3 bucket.
    /// </summary>
    private static async Task VerifyFileExistsAsync()
    {
        _uiMethods.DisplayTitle("Step 5: Verify uploaded file exists");

        Console.WriteLine($"Checking if file exists at {_bucketName}/
{_objectKey}...");

        try
        {
            var metadata = await _s3Wrapper.GetObjectMetadataAsync(_bucketName!,
 _objectKey!);

            Console.WriteLine($"File verification successful! File exists in the
 bucket.");
            Console.WriteLine($"File size: {metadata.ContentLength} bytes");
            Console.WriteLine($"File type: {metadata.Headers.ContentType}");
            Console.WriteLine($"Last modified: {metadata.LastModified}");
        }
        catch (AmazonS3Exception ex) when (ex.StatusCode ==
 System.Net.HttpStatusCode.NotFound)
        {
            Console.WriteLine($"Error: File was not found in the bucket.");
```

```
            throw;
        }
    }

    private static void DisplayPresignedPostFields(CreatePresignedPostResponse
 response)
    {
        Console.WriteLine($"Presigned POST URL: {response.Url}");
        Console.WriteLine("Form fields to include:");

        foreach (var field in response.Fields)
        {
            Console.WriteLine($"  {field.Key}: {field.Value}");
        }
    }

    /// <summary>
    /// Clean up resources created by the scenario.
    /// </summary>
    private static async Task CleanupAsync()
    {
        if (!string.IsNullOrEmpty(_bucketName))
        {
            Console.WriteLine($"Deleting bucket {_bucketName} and its contents...");
            bool result = await _s3Wrapper.DeleteBucketAsync(_bucketName);

            if (result)
            {
                Console.WriteLine("Bucket deleted successfully");
            }
            else
            {
                Console.WriteLine("Failed to delete bucket - it may have been
 already deleted");
            }
        }
    }
}
```

Migrando seu projeto para o AWS SDK para .NET

Esta seção fornece informações sobre tarefas de migração para a AWS SDK para .NET versão 4 do e instruções sobre como realizar essas tarefas. Para obter informações sobre como migrar para versões anteriores (por exemplo, migrar da V2 para a V3 ou de uma subversão da V3 para outra), consulte Migrar seu projeto no Guia do AWS SDK para .NET desenvolvedor (versão 3).

Consulte também O que há de novo e Plataformas compatíveis.

Tópicos

- Migrando para a versão 4 do AWS SDK para .NET
- Migração do .NET Standard 1.3

Migrando para a versão 4 do AWS SDK para .NET

A AWS SDK para .NET versão 4 (V4) tem um número significativo de alterações significativas em relação à versão 3 (V3) do SDK. Este tópico descreve as alterações importantes na versão 4 e o possível trabalho que talvez você precise fazer para migrar seu ambiente ou código da V3. Para obter informações adicionais sobre outras mudanças notáveis no SDK, consulte os seguintes recursos:

- O problema do rastreador de desenvolvimento em:. GitHub https://github.com/aws/aws-sdk-net/issues/3362
- A postagem do blog Prévia 1 da AWS SDK para .NET V4.
- A postagem do blog Preview 4 of AWS SDK para .NET V4.
- A postagem do blog Disponibilidade geral da AWS SDK para .NET V4.0.

NET Framework

O destino.NET Framework 3.5 foi removido da V4 do AWS SDK para .NET. Como resultado, o SDK não é mais compatível com o.NET Framework 3.5. Essa versão do SDK é compilada com o.NET Framework 4.7.2 e é executada no tempo de execução do.NET 4.0. Para obter mais informações, consulte Plataformas suportadas.

Tipos de valor

As propriedades que usam tipos de valor em classes usadas para fazer solicitações e respostas foram alteradas para usar tipos de valores anuláveis. As propriedades com os seguintes tipos foram alteradas:

- `bool`foi alterado para `bool?`
- `double`foi alterado para `double?`
- `int`foi alterado para `int?`
- `float`foi alterado para `float?`
- `long`foi alterado para `long?`
- `Datetime`foi alterado para `Datetime?`

Para obter informações adicionais sobre essa alteração, consulte a postagem do blog Preview 1 of AWS SDK para .NET V4.

Coleções

As propriedades que usam coleções em classes usadas para fazer solicitações e respostas agora usam como padrão`null`. Como resultado, seu código precisa verificar se uma coleção não é nula antes de tentar usá-la. Por exemplo:

```
var sqsClient = new AmazonSQSClient();
var listResponse = await sqsClient.ListQueuesAsync(new ListQueuesRequest());
if (listResponse.QueueUrls != null)
{
    foreach (string qUrl in listResponse.QueueUrls)
    {
        // Perform operations on each queue such as displaying all the attributes.
    }
}
```

O comportamento da V3 de inicializar coleções pode ser restaurado configurando como`Amazon.AWSConfigs.InitializeCollections`. `true` Essa propriedade também existe na V3 para usuários que desejam experimentar essa mudança de comportamento antes de atualizar para a V4.

Para obter informações adicionais sobre essa alteração, consulte a postagem do blog Preview 1 of AWS SDK para .NET V4.

AWS Security Token Service (STS)

- O endpoint regional

  Ao usar provedores de credenciais que dependem AWS STS, as chamadas sempre usam o endpoint regional. Isso difere da V3 do SDK, que usava a `us-east-1` região por padrão ao ser executada na partição pública, independentemente da região configurada.

- O **`StsRegionalEndpointsValue`** enum

  A `StsRegionalEndpointsValue` enumeração foi removida do namespace Amazon.Runtime. Qualquer código usando essa enumeração deve ser removido.

- A classe **`STSAssumeRoleAWSCredentials`**

  O fornecedor obsoleto de credenciais STS assume a função,`STSAssumeRoleAWSCredentials`, foi removido da Amazon. SecurityTokennamespace. Em vez disso, use AssumeRoleAWSCredentialsdo Amazon.Runtime.

Mudanças relacionadas a **`ClientConfig`**

O Amazon.Runtime. ClientConfigclass é a classe base das classes de configuração do cliente de serviço, como AmazonS3Config. As seguintes alterações foram feitas nessa classe base.

- Modo de repetição padrão

  O padrão `RetryMode` da propriedade é em `Standard` vez de. `Legacy` Como resultado, o `Legacy` valor foi removido do Amazon.Runtime. RequestRetryModeenum.

- Modo de configuração padrão

  O padrão `DefaultConfigurationMode` da propriedade é em `Standard` vez de. `Legacy` Como resultado, o `Legacy` valor foi removido do Amazon.Runtime. DefaultConfigurationModeenum.

- A **`ReadWriteTimeout`** propriedade

  A `ReadWriteTimeout` propriedade obsoleta foi removida de todos os destinos, exceto o.NET Framework 4.7.2.

As AWSSDK extensões. NETCore.Pacote de configuração NuGet

As AWSSDKextensões. NETCoreO NuGet pacote.Setup foi atualizado para aliviar os problemas presentes na V3 do SDK, bem como para tornar o pacote seguro para o AOT nativo. Essas mudanças estão resumidas abaixo. Para obter informações detalhadas, consulte PR 3353 no aws-sdk-netrepositório em. GitHub

- A classe **`DefaultClientConfig`**

  A `DefaultClientConfig` classe não é mais herdada da classe base de configuração do cliente de serviço Amazon.Runtime. ClientConfig. As propriedades relevantes de `ClientConfig` foram replicadas `DefaultClientConfig` usando tipos de valores anuláveis. Essa alteração nos permite detectar quando um valor foi definido `DefaultClientConfig` ao copiar os valores para a configuração que está sendo criada para o cliente do serviço.

  Um resultado específico dessa mudança é que não `DefaultClientConfig.HttpClientFactory` está mais disponível na V4. Use `AWSConfigs.HttpClientFactory` em vez disso. Para obter informações adicionais, consulte a GitHub edição 3790.

  Outro resultado dessa mudança é que as sobrecargas genéricas do método de `IConfiguration.GetAWSOptions` extensão que aceitava um objeto de configuração de serviço foram removidas. Em vez disso, a sobrecarga não genérica deve ser usada, e o SDK processará automaticamente o preenchimento das configurações específicas do serviço. Para obter informações adicionais, consulte a GitHub edição 3866.

- AOT nativo

  Um novo mecanismo para criar clientes de serviço que usam métodos de interface estática do C# 11 foi adicionado ao pacote. Essa alteração elimina a necessidade de fazer cargas do tipo Assembly para criar instâncias de clientes de serviço, incluindo a manipulação de strings do nome da interface de serviço para calcular o tipo de cliente de serviço, o que é incompatível com o AOT nativo. Essa alteração está disponível somente para o.NET 8 e versões posteriores; versões mais antigas ainda usam o mecanismo original.

Para obter informações adicionais neste guia sobre esse pacote, consulteAWSSDK.Extensões. NETCore.Configuração e IConfiguration. O código-fonte desse pacote está GitHub emhttps://github.com/aws/aws-sdk-net/tree/main/extensions/src/AWSSDK.Extensions.NETCore.Setup.

`CookieSigner` e `UrlSigner`

As `UrlSigner` extensões `CookieSigner` e da Amazon CloudFront foram movidas para um pacote de extensão separado chamado AWSSDK.Extensions. CloudFront.Signatários. Essa alteração é para oferecer suporte ao OpenSSL 3 e depender do .Cryptography. BouncyCastle

O código-fonte desse pacote está GitHub emhttps://github.com/aws/aws-sdk-net/tree/main/extensions/src/AWSSDK.Extensions.CloudFront.Signers.

DateTime versus UTC DateTime

Algumas classes V3 têm uma DateTime propriedade marcada como “obsoleta” ou “obsoleta”, bem como uma propriedade UTC alternativa. DateTime Nessas classes, a DateTime propriedade obsoleta foi removida e o nome da DateTime propriedade UTC foi alterado para o nome original da DateTime propriedade.

A seguir estão alguns exemplos de classes para as quais essa alteração foi implementada.

- DescribeSpotPriceHistoryRequest:
  - A `StartTime` propriedade obsoleta foi removida e o nome da `StartTimeUtc` propriedade foi alterado para "StartTime”.
  - A `EndTime` propriedade obsoleta foi removida e o nome da `EndTimeUtc` propriedade foi alterado para "EndTime”.
- CreateFleetRequest
  - A `ValidFrom` propriedade obsoleta foi removida e o nome da `ValidFromUtc` propriedade foi alterado para "ValidFrom”.
  - A `ValidUntil` propriedade obsoleta foi removida e o nome da `ValidUntilUtc` propriedade foi alterado para "ValidUntil”.

Essa alteração pode levar a tempos de compensação se um aplicativo estiver usando a DateTime propriedade original e obsoleta. Ocorrerá um erro de tempo de compilação para o código que usa a propriedade UTC DateTime .

DateTime analisando

A DateTimeUnmarshaller aula foi atualizada. Essa classe estava analisando e retornando DateTime strings no horário local. Em alguns casos, esses valores estavam sendo convertidos

novamente para UTC devido a uma atualização anterior, mas nem sempre. Agora, DateTime as cadeias de caracteres que não estão agrupadas são consideradas UTC e serão especificadas e desempacotadas como UTC. Essa atualização inclui as seguintes mudanças de comportamento.

Certas propriedades de timestamp baseadas na DateTime classe estavam sendo analisadas em horários locais. Isso incluía desorganizadores de resposta para carimbos de data/hora e listas de carimbos de data/hora para formatos e. `TimestampFormat.ISO8601` `TimestampFormat.RFC822` DateTime a análise foi atualizada para retornar os horários UTC em vez disso.

`ConvertFromUnixEpochSeconds` e `ConvertFromUnixEpochMilliseconds`

Os ConvertFromUnixEpochMillisecondsmétodos ConvertFromUnixEpochSecondse, que convertem segundos da época Unix em uma DateTime estrutura, estavam retornando a hora do Unix Epoch como uma hora local em vez de uma hora UTC. Esses métodos agora retornam a hora UTC.

Registro em log

A forma como você ativa o login no SDK foi atualizada para a V4. O registro no console e no diagnóstico do sistema funciona da mesma forma que o V3; ou seja, definindo a `LoggingConfig.LogTo` propriedade da AWSConfigsclasse como ou`LoggingOptions.Console`. `LoggingOptions.SystemDiagnostics` A LoggingOptionsopção for `log4net` foi removida junto com a lógica interna do SDK para usar a reflexão para se conectar a uma instância na memória do. `log4net`

Para incluir o registro do SDK em uma estrutura de registro, um pacote de adaptador separado é usado para conectar o SDK à estrutura de registro. Use o pacote AWSSDK.Extensions.Logging.Log4 para e o NetAdaptor .Extensions.Logging. `log4net` AWSSDK ILoggerPacote de adaptador para`Microsoft.Extensions.Logging`. Os exemplos de código a seguir mostram como configurar o registro em log nesses dois casos.

Exemplo de configuração para log4net

Adicione o `AWSSDK.Extensions.Logging.Log4NetAdaptor` NuGet pacote e chame o `ConfigureAWSSDKLogging` método estático de`Log4NetAWSExtensions`.

```
using Amazon.DynamoDBv2;
using Amazon.Extensions.Logging.Log4NetAdaptor;
```

```
using log4net;

[assembly: log4net.Config.XmlConfigurator(ConfigFile = "log4net.config")]

Log4NetAWSExtensions.ConfigureAWSSDKLogging();
var logger = LogManager.GetLogger(typeof(Program));
```

Exemplo de configuração para Microsoft.Extensions.Logging

Adicione o `AWSSDK.Extensions.Logging.ILoggerAdaptor` NuGet pacote e chame o método de `ConfigureAWSSDKLogging` extensão na `ILoggerFactory` interface.

```
var builder = WebApplication.CreateBuilder(args);

var app = builder.Build();

app.Services.GetRequiredService<ILoggerFactory>()
    .ConfigureAWSSDKLogging();
```

Support para HTTP 2

O suporte para HTTP 2 foi adicionado para permitir o streaming bidirecional. Para obter mais informações, consulte Support para HTTP 2.

Autenticação única

O valor padrão da `SupportsGettingNewToken` propriedade da classe SSOAWSCredentialsOptions foi alterado de `true` para`false`. Se você tiver aplicativos que usam a SSOAWSCredentialsclasse para obter credenciais de SSO, talvez seja necessário definir a `Options.SupportsGettingNewToken` propriedade como. `true` Para ver exemplos dessa configuração, consulte os exemplos de código emTutorial para SSO usando somente aplicativos .NET. Para obter informações adicionais, consulte PR 3737 no aws-sdk-net GitHub repositório.

Alterações específicas do DynamoDB

As alterações a seguir são específicas do Amazon DynamoDB. Muitos deles são mudanças significativas.

Para obter informações adicionais sobre mudanças no DynamoDB na V4 do, consulte a postagem AWS SDK para .NET do blog Preview 4 of V4. AWS SDK para .NET

Abrir para ver itens

As mudanças da V4 no SDK para DynamoDB abordam alguns problemas relacionados à testabilidade, mas se concentram principalmente nas bibliotecas de alto nível:

- O modelo de documento.NET, nomeado DocumentModelno código.
- O modelo de persistência de objetos.NET, nomeado DataModelno código.

Para obter informações detalhadas sobre esses modos de programação, consulte DynamoDB este guia.

Modelo de documento: exceção atualizada para interface simulada **`IAmazonDynamoDB`**

No modelo de documento anterior à V4 do SDK, se uma tabela fosse inicializada com uma interface simulada do DynamoDBIAmazon, ela retornaria. `NullReferenceException` Em vez disso, a V4 do SDK retorna`InvalidOperationException`. `Table`Os métodos assíncronos devem funcionar com um cliente simulado, mas você ainda pode ver exceções ao chamar métodos síncronos de. `.NET/Core/Standard`

Para obter mais informações sobre essa alteração, consulte PR 3388 em. GitHub

Modelo de documento: **`FromJson`** e **`ToJson`** métodos

Os `ToJson` métodos `FromJson` e da classe Document agora são usados `System.Text.Json` em vez de LitJson para serialização e LitJson foram removidos da V4 do SDK. Uma vantagem do uso `System.Text.Json` é que esse analisador suporta o uso do `Decimal` tipo.NET, que oferece maior precisão para propriedades numéricas de ponto flutuante.

Modelo de persistência de objetos: a classe **`DynamoDBOperationConfig`**

No modelo de persistência de objetos, as seguintes alterações foram feitas na classe compartilhada do Dynamo Config: DBOperation

- A classe foi separada em novas classes específicas da operação SaveConfig, como, e. LoadConfigQueryConfig Os métodos usados `DynamoDBOperationConfig` foram marcados como obsoletos e estão sujeitos a remoção no futuro.

  Para obter mais informações sobre essa alteração, consulte PR 3421 em GitHub.
- As `DisableFetchingTableMetadata` propriedades `MetadataCachingMode` e foram removidas da classe. Essas propriedades não foram incluídas nas novas classes específicas

da operação mencionadas anteriormente. As propriedades removidas são configurações em nível de tabela que devem ser especificadas na `Context` propriedade global da classe AWSConfigsDynamoDB ou na classe Dynamo Config. DBContext

Para obter mais informações sobre essa alteração, consulte PR 3422 em GitHub.

- A classe não é mais herdada da classe do Dynamo Config DBContext. Isso impede que você passe um `DynamoDBOperationConfig` objeto para o construtor do Dynamo DBContext, onde algumas propriedades na configuração específica da operação (como) não se aplicam. `OverrideTableName`

  Para obter mais informações sobre essa alteração, consulte PR 3422 em GitHub.

Modelo de persistência de objetos: polimorfismo

A DBPolymorphic TypeAttribute classe Dynamo foi adicionada ao modelo de persistência de objetos. Essa classe permite suporte para serialização e desserialização de tipos polimórficos. Para obter mais informações, consulte PR 3643 em. GitHub

Modelo de documento e modelo de persistência de objetos: operações simuláveis

Novas interfaces específicas de operação foram adicionadas para permitir que os clientes simulem as operações do DynamoDB. Os métodos de fábrica na IDynamoDBContextinterface foram atualizados para retornar as novas interfaces.

Para obter mais informações sobre essa alteração, consulte PR 3450 on GitHub.

- Modelo de persistência de objeto
  - `BatchGet`Operações simuladas por meio das `IMultiTableBatchGet` interfaces `IBatchGet` e.
  - `BatchWrite`Operações simuladas por meio das `IMultiTableBatchWrite` interfaces `IBatchWrite` e.
  - `TransactGet`Operações simuladas por meio das `IMultiTableTransactGet` interfaces `ITransactGet` e.
  - `TransactWrite`Operações simuladas por meio das `IMultiTableTransactWrite` interfaces `ITransactWrite` e.
  - Simulação `Scan` e `Query` operações por meio da `IAsyncSearch` interface.
- Modelo de documento

- `Table`Operações simuladas por meio da `ITable` interface.
- Simulação `Scan` e `Query` operações por meio da `ISearch` interface.
- `TransactWrite`Operações simuladas por meio das `IMultiTableDocumentTransactWrite` interfaces `IDocumentTransactWrite` e.
- `TransactGet`Operações simuladas por meio das `IMultiTableDocumentTransactGet` interfaces `IDocumentTransactGet` e.
- `BatchWrite`Operações simuladas por meio das `IMultiTableDocumentBatchWrite` interfaces `IDocumentBatchWrite` e.
- `BatchGet`Operações simuladas por meio das `IMultiTableDocumentBatchGet` interfaces `IDocumentBatchGet` e.

Modelo de documento e modelo de persistência de objetos: Support for native AOT

Uma limitação do AOT nativo é o suporte para tipos.NET aninhados. Em alguns casos, esses tipos aninhados podem passar despercebidos pelo componente de corte do compilador.NET. Nesse caso, você pode receber uma exceção, como: "`System.InvalidOperationException: Type <type> is unsupported, it cannot be instantiated.`"

Você pode contornar essa limitação adicionando `DynamicDependency` algum lugar no caminho do código que informa ao trimmer sobre a dependência do subtipo. O construtor do tipo.NET de nível superior que está sendo salvo é um lugar provável. O exemplo de código a seguir mostra como usar o `DynamicDependency` atributo:

```
[DynamoDBTable("TestTable")]
class TypeWithNestedTypeProperty
{
    [DynamicDependency(DynamicallyAccessedMemberTypes.All, typeof(SubType))]
    public TypeWithNestedTypeProperty()
    {

    }

    [DynamoDBHashKey]
    public string Id { get; set; }
    public string Name { get; set; }

    public SubType SubType { get; set; }
}
```

```
class SubType
{
    public string SubName { get; set; }
}
```

Dínamo DBStreams

O Dynamo DBStreams foi removido do pacote AWSSDK.DynamoDB e está disponível em seu próprio NuGet pacote, AWSSDK.Dynamo, e tem seu próprio DBStreams namespace,. `Amazon.DynamoDBStreams`

Permitir a remoção do **`TableNamePrefix`** valor

Agora você pode remover o valor da `TableNamePrefix` propriedade na classe Dynamo DBContext Config no nível de operação individual. Para obter mais informações sobre essa alteração, consulte PR 3476 em GitHub.

a propriedade **`RetrieveDateTimeInUtc`**

Para a classe Dynamo DBContext Config, o valor padrão `RetrieveDateTimeInUtc` da propriedade foi alterado para. `true`

a propriedade **`DynamoDBContextTableNamePrefix`**

A `DynamoDBContextTableNamePrefix` propriedade foi removida da classe AWSConfigsDynamoDB. Os usuários devem ligar `AWSConfigsDynamoDB.Context.TableNamePrefix` em vez de

Alterações específicas do EC2

As seguintes alterações são específicas do Amazon EC2. A maioria ou todas elas são mudanças significativas.

Abrir para ver itens

`GetDecryptedPassword`

A `GetDecryptedPassword` extensão do Amazon EC2 foi movida para um pacote de extensão separado chamado AWSSDK .extensions.EC2. DecryptPassword. Essa alteração é para oferecer suporte ao OpenSSL 3 e depender do .Cryptography. BouncyCastle

O código-fonte desse pacote está GitHub emhttps://github.com/aws/aws-sdk-net/tree/main/extensions/src/AWSSDK.Extensions.EC2.DecryptPassword.

Support para Amazon EC2 IMDSv1

O suporte para o Instance Metadata Service versão 1 (IMDSv1) foi removido. A V4 do SDK sempre usa o Instance Metadata Service Version 2 (IMDSv2) ao buscar credenciais e outros metadados do IMDS. Para obter mais informações sobre o IMDS, consulte Usar o IMDS no Guia do usuário do Amazon EC2.

Elementos de programação que foram alterados ou removidos

- Todo o `Amazon.EC2.Import` namespace e o código foram removidos.
- Todo o `Amazon.EC2.Util` namespace e o código foram removidos, o que inclui os utilitários da AMI usados para pesquisar o AMIs EC2 para Windows.
- A `IpRanges` propriedade obsoleta foi removida da IpPermissionclasse. Em vez disso, use `Ipv6Ranges` as propriedades `Ipv4Ranges` ou.
- Os seguintes campos obsoletos foram removidos da EC2InstanceMetadataclasse: `EC2_METADATA_SVCEC2_METADATA_ROOT`, `EC2_USERDATA_ROOTEC2_DYNAMICDATA_ROOT`,, `EC2_APITOKEN_URL` e.

Alterações específicas do S3

As seguintes alterações são específicas do Amazon S3. A maioria ou todas elas são mudanças significativas.

Abrir para ver itens

Região da AWS us-east-1

Os clientes do serviço Amazon S3 configurados para a `us-east-1` região não podem mais acessar buckets em outras regiões. Os buckets devem ser acessados com clientes de serviço do S3 configurados para a região em que o bucket está.

Para obter informações adicionais sobre essa alteração, consulte a postagem do blog Preview 4 of AWS SDK para .NET V4.

Cliente de criptografia S3

O cliente de criptografia Amazon S3, definido no `Amazon.S3.Encryption` namespace, foi removido do pacote.S3. AWSSDK Esse cliente foi movido para seu próprio pacote chamado Amazon.Extensions.S3.Encryption, e a documentação para ele está em. https://aws.github.io/amazon-s3-encryption-client-dotnet/api/Amazon.Extensions.S3.Encryption.html Para obter

informações sobre migração, consulte the section called “Migração do cliente de criptografia S3 (V1 para V2)”. Para obter mais informações sobre a criptografia S3, consulte Algoritmos de criptografia compatíveis no Guia do desenvolvedor do Amazon S3 Encryption Client.

Diretiva de marcação S3 para **`CopyObject`**

A `TaggingDirective` propriedade foi exposta como uma propriedade pública da CopyObjectRequestclasse, que é usada por `AmazonS3Client.CopyObject` métodos. Essa propriedade corresponde ao `x-amz-tagging-directive` parâmetro Amazon S3, conforme definido na CopyObjectação.

A diretiva de marcação não é mais definida automaticamente como COPY. Se um desenvolvedor não especificar uma diretiva de marcação, o back-end do S3 assumirá automaticamente que é COPY, mas se um desenvolvedor definir explicitamente a propriedade como nula, o valor não será definido de forma alguma.

A **`UseArnRegion`** propriedade para a configuração do S3

A `UseArnRegion` propriedade da classe Amazon.s3.amazons3Config foi atualizada para que a variável de `AWS_S3_USE_ARN_REGION` ambiente tenha precedência sobre a configuração no arquivo compartilhado. `s3_use_arn_region` AWS `config` Para obter mais informações sobre essas variáveis e configurações, consulte Referência de configurações no Guia AWS SDKs de referência de ferramentas.

Barras iniciais para os métodos **`CopyObject`** e **`CopyPart`**

As barras iniciais não serão mais cortadas para o Amazon `CopyObject` `CopyPart` S3 e os métodos. A `DisableTrimmingLeadingSlash` propriedade foi removida das CopyPartRequestclasses CopyObjectRequeste.

Os **`DoesS3BucketExist...`** métodos

O obsoleto `DoesS3BucketExist` e `DoesS3BucketExistAsync` os métodos foram removidos da classe AmazonS3Util, que implementa a interface AmazonS3. ICore Esses métodos foram removidos porque sempre usam HTTP. Em vez disso, use DoeSS3 BucketExist V2 e BucketExistDoesS3 V2Async.

O SDK sempre usa SigV4

A versão 4 do AWS SDK para .NET sempre usa a AWS Signature Version 4 (SigV4) para assinar solicitações. Essa alteração resulta nas seguintes alterações relacionadas:

- A `UseSignatureVersion4` propriedade da classe AWSConfigsS3 foi removida.
- A `SignatureVersion` propriedade do Amazon.Runtime. ClientConfiga classe foi removida. Essa propriedade foi usada somente pelo Amazon S3 para compatibilidade com versões anteriores.
- A `RegionEndpoint.Endpoint` classe foi removida. Isso inclui a `SignatureVersionOverride` propriedade, que foi usada para substituir as versões de assinatura do Amazon S3. Em vez disso, use o `client.DetermineServiceOperationEndPoint()` método específico do serviço.
- Métodos atualizados Amazons3util. PostUploade S3PostUploadSignedPolicy. GetSignedPolicypara usar o SigV4. Como consequência, o `S3PostUploadSignedPolicy.GetSignedPolicyV4` método foi removido porque `GetSignedPolicy` agora executa a mesma função. Além disso, `GetSignedPolicy` foi fornecido um terceiro parâmetro para o endpoint da região.

Os **`PutACL`** métodos **`GetACL`** e

Os `PutACL` métodos `GetACL` e da classe Amazons3Client foram marcados como obsoletos. Para acessar a funcionalidade desses métodos, use os seguintes novos métodos em vez disso: `GetBucketACL` `PutBucketACLGetObjectACL`,, `PutObjectACL` e.

Elementos de programação obsoletos removidos

Vários elementos de programação da implementação do Amazon S3 foram removidos da V4 do SDK, incluindo valores de enumeração, tipos, métodos, namespaces etc. Eles estão listados abaixo, se ainda não foram abordados anteriormente, junto com as possíveis etapas que você pode tomar para acomodar sua remoção.

- A `DisableMD5Stream` propriedade foi removida da TransferUtilityUploadRequestclasse. Use a propriedade `DisableDefaultChecksumValidation`.

  Além disso, a `CalculateContentMD5Header` propriedade foi removida da `TransferUtilityUploadRequest` classe. Essa propriedade não é mais necessária porque o SDK calcula uma soma de verificação por padrão.
- As `ServerSideEncryptionKeyManagementServiceKeyId` propriedades `ServerSideEncryptionMethod` e foram removidas da CopyPartRequestclasse. Em vez disso, use as propriedades com os mesmos nomes na InitiateMultipartUploadRequestclasse, que são usadas em alguns dos `InitiateMultipartUpload...` métodos da classe AmazonS3Client.

- A `Expires` propriedade foi removida da GetObjectResponseclasse. Use a propriedade `ExpiresString`. A string pode não estar em um formato de carimbo de data/hora válido, então seu código deve usar o `TryParse` método ao converter em a. `DateTime`
- Região da AWS Identificadores obsoletos foram removidos da enumeração S3Region.
- A `Prefix` propriedade foi removida da LifecycleRuleclasse. Use a propriedade `Filter`.

  Além disso, as `Transition` propriedades `NoncurrentVersionTransition` e foram removidas da `LifecycleRule` classe. Em vez dissoNoncurrentVersionTransitions, use `Transitions` as coleções e.
- A `Event` propriedade foi removida da TopicConfigurationclasse. Em vez disso, use a `Events` coleção.
- CalculateContentMD5Propriedade do cabeçalho. Essa propriedade não precisa mais ser definida porque o SDK computará uma soma de verificação por padrão.
- A `Bucket` propriedade foi removida da SelectObjectContentRequestclasse. Use a propriedade `BucketName`.
- A `NumberOfUploadThreads` propriedade foi removida da TransferUtilityConfigclasse. Em vez disso, use a `ConcurrentServiceRequests` propriedade.

Elementos de programação que foram removidos

Vários elementos de programação foram removidos da V4 do SDK, incluindo valores de enumeração, tipos, métodos, namespaces etc. Eles estão listados abaixo, se ainda não foram abordados anteriormente, junto com as possíveis etapas que você pode tomar para acomodar sua remoção.

Abrir para ver itens

O namespace **`Amazon.Auth.AccessControlPolicy.ActionIdentifiers`**.

O `Amazon.Auth.AccessControlPolicy.ActionIdentifiers` namespace foi removido. Isso inclui identificadores de ação do IAM, que foram definidos na `IdentityandAccessManagementActionIdentifiers` classe. O código que usa esses identificadores de ação deve ser alterado para usar valores de string do nome da ação.

Para obter mais informações, consulte the section called “Criar políticas gerenciadas a partir do JSON” a Visão geral das políticas de JSON no Guia do usuário do IAM.

A classe **`ClientConfig`**

O Amazon.Runtime. ClientConfigclass é a classe base das classes de configuração do cliente de serviço, como AmazonS3Config. Os seguintes elementos de programação foram removidos dessa classe.

- Os `DetermineDnsSuffix` métodos `DetermineServiceURL` e foram removidos. Em vez disso, use o `DetermineServiceOperationEndpoint` método do cliente de serviço; por exemplo, AmazonS3Client. DetermineServiceOperationEndpoint.
- A `ReadEntireResponse` propriedade foi removida. Em vez disso, use uma das seguintes opções:
  - A `LogResponses` propriedade do AWSConfigs. LoggingConfigclasse.
  - A `LogResponse` propriedade da configuração do cliente; por exemplo, AmazonS3Config.

O namespace **`Amazon.Runtime`**.

O namespace Amazon.Runtime foi atualizado da seguinte forma:

- A `ECSTaskCredentials` classe obsoleta foi removida do namespace. Em vez disso, use o GenericContainerCredentialsprovedor, que também oferece suporte ao Amazon EKS Pod Identities.
- O obsoleto `StoredProfileAWSCredentials` e `StoredProfileCredentials` as classes foram removidos do namespace. Use o Net SDKCredentials File ou a SharedCredentialsFileclasse do Amazon.Runtime. CredentialManagementem vez disso, namespace.
- O `HasCachedAccessTokenAvailable` método obsoleto da SSOAWSCredentialsclasse foi removido do namespace.
- A `EnvironmentAWSCredentials` classe obsoleta foi removida do namespace. No lugar, use a classe AppConfigAWSCredentials.
- A `StoredProfileFederatedCredentials` classe obsoleta foi removida do namespace. Em vez disso, use a AWSCredentials classe Federated.
- As seguintes classes obsoletas foram removidas do namespace: `EnvironmentVariableAWSEndpointDiscoveryEnabledProfileAWSEndpointDiscoveryEnable` e. `FallbackEndpointDiscoveryEnabledFactory`
- A `UseSigV4` propriedade obsoleta foi removida da AmazonWebServiceRequestclasse. Use a propriedade `SignatureVersion`.

- A `ProfileIniFile` classe no `Amazon.Runtime.Internal.Util` namespace tem um método sobrecarregado chamado. `TryGetSection` As versões do método que não suportam o `out` parâmetro for `nestedProperties` foram removidas da classe.
- A `EventBridgeSigner` classe obsoleta no `Amazon.Runtime.Internal.Auth` namespace foi removida.
- A propriedade obsoleta do `Parameters` dicionário foi removida da WebServiceRequestEventArgsclasse. Use a propriedade `ParameteCollection`.

BouncyCastle

A cópia de origem do BouncyCastle foi removida da V4 do SDK.

A classe **`StoredProfileSAMLCredentials`**

A `StoredProfileSAMLCredentials` classe obsoleta na Amazon. SecurityTokenO namespace.SAML foi removido. Em vez disso, use a AWSCredentials classe Federated no namespace Amazon.Runtime.

A classe **`AWSSDKUtils`**

Os métodos a seguir foram removidos da AWSSDKUtilsclasse: `ResolveResourcePathProtectEncodedSlashUrlEncode`, `ConvertToUnixEpochMilliSeconds` e.

A classe **`ProfileManager`**

A `ProfileManager` classe obsoleta foi removida do namespace Amazon.Util. Use o SDKCredentialsarquivo de rede ou a SharedCredentialsFileclasse do Amazon.Runtime. CredentialManagementem vez disso, namespace.

A classe **`AWSConfigs`**

As seguintes propriedades obsoletas foram removidas da AWSConfigsclasse: `LoggingResponseLogging`, `LogMetrics` e. Use a propriedade `LoggingConfig`.

A classe **`ConditionFactory`**

O método com a seguinte assinatura foi removido da ConditionFactoryclasse:`NewCondition(ConditionFactory.DateComparisonType, DateTime)`. Em vez disso, use o NewConditionUtcmétodo.

CloudFront Utilitários da Amazon

O `Amazon.CloudFront.Util` namespace e a `AmazonCloudFrontUtil` classe obsoletos foram removidos.

AWS IoT

Na ListPrincipalThingsResponseclasse, uma personalização antiga para uma `NextToken` substituição foi removida em favor da paginação.

AWS Lambda

Os seguintes `Invoke...` métodos da AmazonLambdaClientclasse foram removidos porque os nomes eram confusos.

- O método V3 com a seguinte assinatura foi removido:`InvokeAsyncResponse InvokeAsync(InvokeAsyncRequest)`. Esse é um método síncrono na V3 do SDK. Em vez disso, use `InvokeResponse Invoke(InvokeRequest)` (para processamento síncrono) ou `Task InvokeAsync(InvokeRequest, CancellationToken)` (para processamento assíncrono).
- O método V3 com a seguinte assinatura foi removido:`Task InvokeAsyncAsync(InvokeAsyncRequest, CancellationToken)`. Esse é um método assíncrono na V3 do SDK. Use `Task InvokeAsync(InvokeRequest, CancellationToken)` em vez disso.

Amazon SageMaker Runtime

Os construtores obsoletos da PayloadPartclasse foram removidos.

Migração do .NET Standard 1.3

Em 27 de junho de 2019, a Microsoft encerrou o suporte para as versões .NET Core 1.0 e .NET Core 1.1. Após esse anúncio, AWS encerrou o suporte do.NET Standard 1.3 AWS SDK para .NET em 31 de dezembro de 2020.

AWS continuou a fornecer atualizações de serviços e correções de segurança para o.NET Standard 1.3 AWS SDK para .NET de destino até 1º de outubro de 2020. Depois dessa data, o alvo do.NET Standard 1.3 entrou no modo de manutenção, o que significa que nenhuma nova atualização foi lançada; AWS aplicou somente correções críticas de bugs e patches de segurança.

Em 31 de dezembro de 2020, o suporte para o.NET Standard 1.3 no AWS SDK para .NET chegou ao fim. Após esta data, nenhuma correção de bugs ou patches de segurança foi aplicada. Os artefatos criados com esse alvo permanecem disponíveis para download em. NuGet

O que você precisa fazer

- Se estiver executando aplicativos usando .NET Framework, você não será afetado.
- Se você estiver executando aplicativos usando .NET Core 2.0 ou superior, você não será afetado.
- Se estiver executando aplicativos usando .NET Core 1.0 ou .NET Core 1.1, faça a migração dos seus aplicativos para uma versão mais recente do .NET Core seguindo as instruções de migração da Microsoft. Recomendamos no mínimo .NET Core 3.1.
- Se estiver executando aplicativos críticos para os negócios que não podem ser atualizados neste momento, você poderá continuar usando sua versão atual do AWS SDK para .NET.

Se tiver dúvidas ou perguntas, entre em contato com o AWS Support.

Segurança para este AWS produto ou serviço

A segurança da nuvem na Amazon Web Services (AWS) é a nossa maior prioridade. Como cliente da AWS , você contará com um data center e uma arquitetura de rede criados para atender aos requisitos das organizações com as maiores exigências de segurança. A segurança é uma responsabilidade compartilhada entre você AWS e você. O modelo de responsabilidade compartilhada descreve isso como a Segurança da nuvem e a Segurança na nuvem.

Segurança da nuvem — AWS é responsável por proteger a infraestrutura que executa todos os serviços oferecidos na AWS nuvem e fornecer serviços que você possa usar com segurança. Nossa responsabilidade de segurança é a maior prioridade em AWS, e a eficácia de nossa segurança é regularmente testada e verificada por auditores terceirizados como parte dos Programas de AWS Conformidade.

Segurança na nuvem — Sua responsabilidade é determinada pelo AWS serviço que você está usando e por outros fatores, incluindo a sensibilidade de seus dados, os requisitos da sua organização e as leis e regulamentos aplicáveis.

Esse AWS produto ou serviço segue o modelo de responsabilidade compartilhada por meio dos serviços específicos da Amazon Web Services (AWS) que ele suporta. Para AWS obter informações sobre segurança do AWS serviço, consulte a página de documentação de segurança do serviço e os AWS serviços que estão no escopo dos esforços de AWS conformidade do programa de conformidade.

Tópicos

- Proteção de dados neste AWS produto ou serviço
- Gerenciamento de Identidade e Acesso
- Validação de conformidade para este AWS produto ou serviço
- Resiliência para este AWS produto ou serviço
- Segurança da infraestrutura para este AWS produto ou serviço
- Aplicando uma versão mínima do TLS no AWS SDK para .NET
- Migração do cliente de criptografia S3 (V1 para V2)
- Migração do cliente de criptografia S3 (V2 para V4)

Proteção de dados neste AWS produto ou serviço

O modelo de responsabilidade AWS compartilhada de se aplica à proteção de dados neste AWS produto ou serviço. Conforme descrito neste modelo, AWS é responsável por proteger a infraestrutura global que executa todos os Nuvem AWS. Você é responsável por manter o controle sobre o conteúdo hospedado nessa infraestrutura. Você também é responsável pelas tarefas de configuração e gerenciamento de segurança dos Serviços da AWS que usa. Para saber mais sobre a privacidade de dados, consulte as Data Privacy FAQ. Para saber mais sobre a proteção de dados na Europa, consulte a postagem do blog AWS Shared Responsibility Model and RGPD no Blog de segurança da AWS .

Para fins de proteção de dados, recomendamos que você proteja Conta da AWS as credenciais e configure usuários individuais com Centro de Identidade do AWS IAM ou AWS Identity and Access Management (IAM). Dessa maneira, cada usuário receberá apenas as permissões necessárias para cumprir suas obrigações de trabalho. Recomendamos também que você proteja seus dados das seguintes formas:

- Use uma autenticação multifator (MFA) com cada conta.
- Use SSL/TLS para se comunicar com AWS os recursos. Exigimos TLS 1.2 e recomendamos TLS 1.3.
- Configure a API e o registro de atividades do usuário com AWS CloudTrail. Para obter informações sobre o uso de CloudTrail trilhas para capturar AWS atividades, consulte Como trabalhar com CloudTrail trilhas no Guia AWS CloudTrail do usuário.
- Use soluções de AWS criptografia, juntamente com todos os controles de segurança padrão Serviços da AWS.
- Use serviços gerenciados de segurança avançada, como o Amazon Macie, que ajuda a localizar e proteger dados sensíveis armazenados no Amazon S3.
- Se você precisar de módulos criptográficos validados pelo FIPS 140-3 ao acessar AWS por meio de uma interface de linha de comando ou de uma API, use um endpoint FIPS. Para saber mais sobre os endpoints FIPS disponíveis, consulte Federal Information Processing Standard (FIPS) 140-3.

É altamente recomendável que nunca sejam colocadas informações confidenciais ou sensíveis, como endereços de e-mail de clientes, em tags ou campos de formato livre, como um campo Nome. Isso inclui quando você trabalha com este AWS produto, serviço ou outro Serviços da AWS usando o console, a API ou AWS SDKs. AWS CLI Quaisquer dados inseridos em tags ou em campos de texto

de formato livre usados para nomes podem ser usados para logs de faturamento ou de diagnóstico. Se você fornecer um URL para um servidor externo, é fortemente recomendável que não sejam incluídas informações de credenciais no URL para validar a solicitação nesse servidor.

Gerenciamento de Identidade e Acesso

AWS Identity and Access Management (IAM) é uma ferramenta AWS service (Serviço da AWS) que ajuda o administrador a controlar com segurança o acesso aos AWS recursos. Os administradores do IAM controlam quem pode ser autenticado (conectado) e autorizado (tem permissões) a usar AWS os recursos. O IAM é um AWS service (Serviço da AWS) que você pode usar sem custo adicional.

Tópicos

- Público
- Autenticação com identidades
- Gerenciar o acesso usando políticas
- Como Serviços da AWS trabalhar com o IAM
- Solução de problemas AWS de identidade e acesso

Público

A forma como você usa AWS Identity and Access Management (IAM) difere, dependendo do trabalho que você faz AWS.

Usuário do serviço — Se você Serviços da AWS costuma fazer seu trabalho, seu administrador fornece as credenciais e as permissões de que você precisa. À medida que você usa mais AWS recursos para fazer seu trabalho, talvez precise de permissões adicionais. Entender como o acesso é gerenciado pode ajudar a solicitar as permissões corretas ao administrador. Se você não conseguir acessar um recurso no AWS, consulte Solução de problemas AWS de identidade e acesso ou o guia do usuário do AWS service (Serviço da AWS) que você está usando.

Administrador de serviços — Se você é responsável pelos AWS recursos da sua empresa, provavelmente tem acesso total AWS a. É seu trabalho determinar quais AWS recursos e recursos seus usuários do serviço devem acessar. Envie as solicitações ao administrador do IAM para alterar as permissões dos usuários de serviço. Revise as informações nesta página para compreender os conceitos básicos do IAM. Para saber mais sobre como sua empresa pode usar o IAM com AWS, consulte o guia do usuário do AWS service (Serviço da AWS) que você está usando.

Administrador do IAM: se você for um administrador do IAM, talvez queira saber detalhes sobre como pode gravar políticas para gerenciar o acesso ao AWS. Para ver exemplos de políticas AWS baseadas em identidade que você pode usar no IAM, consulte o guia do usuário do AWS service (Serviço da AWS) que você está usando.

Autenticação com identidades

A autenticação é a forma como você faz login AWS usando suas credenciais de identidade. Você deve estar autenticado como usuário do IAM ou assumindo uma função do IAM. Usuário raiz da conta da AWS

Você pode fazer login como uma identidade federada usando credenciais de uma fonte de identidade como Centro de Identidade do AWS IAM (IAM Identity Center), autenticação de login único ou credenciais. Google/Facebook Para ter mais informações sobre como fazer login, consulte Como fazer login em sua Conta da AWS no Guia do usuário do Início de Sessão da AWS .

Para acesso programático, AWS fornece um SDK e uma CLI para assinar solicitações criptograficamente. Para ter mais informações, consulte AWS Signature Version 4 para solicitações de API no Guia do usuário do IAM.

Conta da AWS usuário root

Ao criar um Conta da AWS, você começa com uma identidade de login chamada usuário Conta da AWS raiz que tem acesso completo a todos Serviços da AWS os recursos. É altamente recomendável não usar o usuário-raiz em tarefas diárias. Consulte as tarefas que exigem credenciais de usuário-raiz em Tarefas que exigem credenciais de usuário-raiz no Guia do usuário do IAM.

Identidade federada

Como prática recomendada, exija que os usuários humanos usem a federação com um provedor de identidade para acessar Serviços da AWS usando credenciais temporárias.

Uma identidade federada é um usuário do seu diretório corporativo, provedor de identidade da web ou Directory Service que acessa Serviços da AWS usando credenciais de uma fonte de identidade. As identidades federadas assumem funções que oferecem credenciais temporárias.

Para o gerenciamento de acesso centralizado, recomendamos Centro de Identidade do AWS IAM. Para saber mais, consulte O que é o IAM Identity Center? no Guia do usuário do Centro de Identidade do AWS IAM .

Usuários e grupos do IAM

Um usuário do IAM é uma identidade com permissões específicas para uma única pessoa ou aplicação. É recomendável usar credenciais temporárias, em vez de usuários do IAM com credenciais de longo prazo. Para obter mais informações, consulte Exigir que usuários humanos usem a federação com um provedor de identidade para acessar AWS usando credenciais temporárias no Guia do usuário do IAM.

Um grupo do IAM especifica um conjunto de usuários do IAM e facilita o gerenciamento de permissões para grandes conjuntos de usuários. Para ter mais informações, consulte Casos de uso de usuários do IAM no Guia do usuário do IAM.

Perfis do IAM

Uma perfil do IAM é uma identidade com permissões específicas que oferece credenciais temporárias. Você pode assumir uma função mudando de um usuário para uma função do IAM (console) ou chamando uma operação de AWS API AWS CLI ou. Para saber mais, consulte Métodos para assumir um perfil no Manual do usuário do IAM.

Os perfis do IAM são úteis para acesso de usuário federado, permissões de usuário do IAM temporárias, acesso entre contas, acesso entre serviços e aplicações em execução no Amazon EC2. Consulte mais informações em Acesso a recursos entre contas no IAM no Guia do usuário do IAM.

Gerenciar o acesso usando políticas

Você controla o acesso AWS criando políticas e anexando-as a AWS identidades ou recursos. Uma política define permissões quando associada a uma identidade ou recurso. AWS avalia essas políticas quando um diretor faz uma solicitação. A maioria das políticas é armazenada AWS como documentos JSON. Para ter mais informações sobre documentos de política JSON, consulte Visão geral das políticas JSON no Guia do usuário do IAM.

Por meio de políticas, os administradores especificam quem tem acesso a que, definindo qual entidade principal pode realizar ações em quais recursos e sob quais condições.

Por padrão, usuários e perfis não têm permissões. Um administrador do IAM cria políticas do IAM e as adiciona aos perfis, os quais os usuários podem então assumir. As políticas do IAM definem permissões, independentemente do método usado para realizar a operação.

Políticas baseadas em identidade

As políticas baseadas em identidade são documentos de políticas de permissão JSON que você anexa a uma identidade (usuário, grupo ou perfil). Essas políticas controlam quais ações as identidades podem realizar, em quais recursos e sob quais condições. Para saber como criar uma política baseada em identidade, consulte Definir permissões personalizadas do IAM com as políticas gerenciadas pelo cliente no Guia do Usuário do IAM.

As políticas baseadas em identidade podem ser políticas em linha (incorporadas diretamente em uma única identidade) ou políticas gerenciadas (políticas autônomas anexadas a várias identidades). Para saber como escolher entre uma política gerenciada e políticas em linha, consulte Escolher entre políticas gerenciadas e políticas em linha no Guia do usuário do IAM.

Políticas baseadas em recursos

Políticas baseadas em recursos são documentos de políticas JSON que você anexa a um recurso. Entre os exemplos estão políticas de confiança de perfil do IAM e políticas de bucket do Amazon S3. Em serviços compatíveis com políticas baseadas em recursos, os administradores de serviço podem usá-las para controlar o acesso a um recurso específico. É necessário especificar uma entidade principal em uma política baseada em recursos.

Políticas baseadas em recursos são políticas em linha localizadas nesse serviço. Você não pode usar políticas AWS gerenciadas do IAM em uma política baseada em recursos.

Listas de controle de acesso (ACLs)

As listas de controle de acesso (ACLs) controlam quais diretores (membros da conta, usuários ou funções) têm permissões para acessar um recurso. ACLs são semelhantes às políticas baseadas em recursos, embora não usem o formato de documento de política JSON.

O Amazon S3 e o AWS WAF Amazon VPC são exemplos de serviços que oferecem suporte. ACLs Para saber mais ACLs, consulte a visão geral da lista de controle de acesso (ACL) no Guia do desenvolvedor do Amazon Simple Storage Service.

Outros tipos de política

AWS oferece suporte a tipos de políticas adicionais que podem definir o máximo de permissões concedidas por tipos de políticas mais comuns:

- Limites de permissões: definem o número máximo de permissões que uma política baseada em identidade pode conceder a uma entidade do IAM. Para saber mais sobre limites de permissões, consulte Limites de permissões para identidades do IAM no Guia do usuário do IAM.
- Políticas de controle de serviço (SCPs) — Especifique as permissões máximas para uma organização ou unidade organizacional em AWS Organizations. Para saber mais, consulte Políticas de controle de serviço no Guia do usuário do AWS Organizations .
- Políticas de controle de recursos (RCPs) — Defina o máximo de permissões disponíveis para recursos em suas contas. Para obter mais informações, consulte Políticas de controle de recursos (RCPs) no Guia AWS Organizations do usuário.
- Políticas de sessão: políticas avançadas transmitidas como um parâmetro durante a criação de uma sessão temporária para um perfil ou um usuário federado. Para saber mais, consulte Políticas de sessão no Guia do usuário do IAM.

Vários tipos de política

Quando vários tipos de política são aplicáveis a uma solicitação, é mais complicado compreender as permissões resultantes. Para saber como AWS determinar se uma solicitação deve ser permitida quando vários tipos de políticas estão envolvidos, consulte Lógica de avaliação de políticas no Guia do usuário do IAM.

Como Serviços da AWS trabalhar com o IAM

Para ter uma visão de alto nível de como Serviços da AWS funciona com a maioria dos recursos do IAM, consulte AWS os serviços que funcionam com o IAM no Guia do usuário do IAM.

Para saber como usar um específico AWS service (Serviço da AWS) com o IAM, consulte a seção de segurança do Guia do usuário do serviço relevante.

Solução de problemas AWS de identidade e acesso

Use as informações a seguir para ajudá-lo a diagnosticar e corrigir problemas comuns que você pode encontrar ao trabalhar com AWS um IAM.

Tópicos

- Não estou autorizado a realizar uma ação em AWS
- Não estou autorizado a realizar iam: PassRole
- Quero permitir que pessoas fora da minha Conta da AWS acessem meus AWS recursos

Não estou autorizado a realizar uma ação em AWS

Se você receber uma mensagem de erro informando que não tem autorização para executar uma ação, suas políticas deverão ser atualizadas para permitir que você realize a ação.

O erro do exemplo a seguir ocorre quando o usuário do IAM `mateojackson` tenta usar o console para visualizar detalhes sobre um atributo *`my-example-widget`* fictício, mas não tem as permissões `awes:`*`GetWidget`* fictícias.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 awes:GetWidget on resource: my-example-widget
```

Nesse caso, a política do usuário `mateojackson` deve ser atualizada para permitir o acesso ao recurso *`my-example-widget`* usando a ação `awes:`*`GetWidget`*.

Se precisar de ajuda, entre em contato com seu AWS administrador. Seu administrador é a pessoa que forneceu suas credenciais de login.

Não estou autorizado a realizar iam: PassRole

Se você receber uma mensagem de erro informando que não está autorizado a executar a ação `iam:PassRole`, as suas políticas devem ser atualizadas para permitir que você passe uma função para o AWS.

Alguns Serviços da AWS permitem que você passe uma função existente para esse serviço em vez de criar uma nova função de serviço ou uma função vinculada ao serviço. Para fazê-lo, você deve ter permissões para passar o perfil para o serviço.

O exemplo de erro a seguir ocorre quando uma usuária do IAM chamada `marymajor` tenta utilizar o console para executar uma ação no AWS. No entanto, a ação exige que o serviço tenha permissões concedidas por um perfil de serviço. Mary não tem permissões para passar o perfil para o serviço.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

Nesse caso, as políticas de Mary devem ser atualizadas para permitir que ela realize a ação `iam:PassRole`.

Se precisar de ajuda, entre em contato com seu AWS administrador. Seu administrador é a pessoa que forneceu suas credenciais de login.

Quero permitir que pessoas fora da minha Conta da AWS acessem meus AWS recursos

É possível criar um perfil que os usuários de outras contas ou pessoas fora da organização podem usar para acessar seus recursos. É possível especificar quem é confiável para assumir o perfil. Para serviços que oferecem suporte a políticas baseadas em recursos ou listas de controle de acesso (ACLs), você pode usar essas políticas para conceder às pessoas acesso aos seus recursos.

Para saber mais, consulte:

- Para saber se é AWS compatível com esses recursos, consulteComo Serviços da AWS trabalhar com o IAM.
- Para saber como fornecer acesso aos seus recursos em todos os Contas da AWS que você possui, consulte Como fornecer acesso a um usuário do IAM em outro Conta da AWS que você possui no Guia do usuário do IAM.
- Para saber como fornecer acesso aos seus recursos a terceiros Contas da AWS, consulte Como fornecer acesso Contas da AWS a terceiros no Guia do usuário do IAM.
- Para saber como conceder acesso por meio da federação de identidades, consulte Conceder acesso a usuários autenticados externamente (federação de identidades) no Guia do usuário do IAM.
- Para saber a diferença entre perfis e políticas baseadas em recurso para acesso entre contas, consulte Acesso a recursos entre contas no IAM no Guia do usuário do IAM.

Validação de conformidade para este AWS produto ou serviço

Para saber se um AWS service (Serviço da AWS) está dentro do escopo de programas de conformidade específicos, consulte Serviços da AWS Escopo por Programa de Conformidade Serviços da AWS e escolha o programa de conformidade em que você está interessado. Para obter informações gerais, consulte Programas de AWS conformidade Programas AWS de .

Você pode baixar relatórios de auditoria de terceiros usando AWS Artifact. Para obter mais informações, consulte Baixar relatórios em AWS Artifact .

Sua responsabilidade de conformidade ao usar Serviços da AWS é determinada pela confidencialidade de seus dados, pelos objetivos de conformidade de sua empresa e pelas leis e regulamentações aplicáveis. Para obter mais informações sobre sua responsabilidade de conformidade ao usar Serviços da AWS, consulte a documentação AWS de segurança.

Esse AWS produto ou serviço segue o modelo de responsabilidade compartilhada por meio dos serviços específicos da Amazon Web Services (AWS) que ele suporta. Para AWS obter informações sobre segurança do AWS serviço, consulte a página de documentação de segurança do serviço e os AWS serviços que estão no escopo dos esforços de AWS conformidade do programa de conformidade.

Resiliência para este AWS produto ou serviço

A infraestrutura AWS global é construída em torno Regiões da AWS de zonas de disponibilidade.

Regiões da AWS fornecem várias zonas de disponibilidade fisicamente separadas e isoladas, conectadas a redes de baixa latência, alta taxa de transferência e alta redundância.

Com as zonas de disponibilidade, é possível projetar e operar aplicações e bancos de dados que automaticamente executam o failover entre as zonas sem interrupção. As zonas de disponibilidade são altamente disponíveis, tolerantes a falhas e escaláveis que uma ou várias infraestruturas de data center tradicionais.

Para obter mais informações sobre AWS regiões e zonas de disponibilidade, consulte Infraestrutura AWS global.

Esse AWS produto ou serviço segue o modelo de responsabilidade compartilhada por meio dos serviços específicos da Amazon Web Services (AWS) que ele suporta. Para AWS obter informações sobre segurança do AWS serviço, consulte a página de documentação de segurança do serviço e os AWS serviços que estão no escopo dos esforços de AWS conformidade do programa de conformidade.

Segurança da infraestrutura para este AWS produto ou serviço

Esse AWS produto ou serviço usa serviços gerenciados e, portanto, é protegido pela segurança de rede AWS global. Para obter informações sobre serviços AWS de segurança e como AWS proteger a infraestrutura, consulte AWS Cloud Security. Para projetar seu AWS ambiente usando as melhores práticas de segurança de infraestrutura, consulte Proteção de infraestrutura no Security Pillar AWS Well-Architected Framework.

Você usa chamadas de API AWS publicadas para acessar este AWS Produto ou Serviço pela rede. Os clientes devem oferecer compatibilidade com:

- Transport Layer Security (TLS). Exigimos TLS 1.2 e recomendamos TLS 1.3.

- Conjuntos de criptografia com perfect forward secrecy (PFS) como DHE (Ephemeral Diffie-Hellman) ou ECDHE (Ephemeral Elliptic Curve Diffie-Hellman). A maioria dos sistemas modernos, como Java 7 e versões posteriores, comporta esses modos.

Além disso, as solicitações devem ser assinadas usando um ID da chave de acesso e uma chave de acesso secreta associada a uma entidade principal do IAM. Ou é possível usar o AWS Security Token Service (AWS STS) para gerar credenciais de segurança temporárias para assinar solicitações.

Esse AWS produto ou serviço segue o modelo de responsabilidade compartilhada por meio dos serviços específicos da Amazon Web Services (AWS) que ele suporta. Para AWS obter informações sobre segurança do AWS serviço, consulte a página de documentação de segurança do serviço e os AWS serviços que estão no escopo dos esforços de AWS conformidade do programa de conformidade.

Aplicando uma versão mínima do TLS no AWS SDK para .NET

Para aumentar a segurança ao se comunicar com AWS serviços, você deve configurar o AWS SDK para .NET para usar o TLS 1.2 ou posterior.

O AWS SDK para .NET usa o tempo de execução do.NET subjacente para determinar qual protocolo de segurança usar. Por padrão, as versões atuais do .NET usam o protocolo configurado mais recentemente que for compatível com o sistema operacional. Seu aplicativo pode substituir esse comportamento do SDK, mas não é recomendado fazê-lo.

.NET Core

Por padrão, o .NET Core usa o protocolo configurado mais recentemente que for compatível com o sistema operacional. O AWS SDK para .NET não fornece um mecanismo para substituir isso.

Se você estiver usando uma versão do .NET Core anterior à 2.1, recomendamos veementemente que atualize sua versão do .NET Core.

Consulte os itens a seguir para obter informações específicas sobre cada sistema operacional.

Windows

As distribuições modernas do Windows têm o suporte para TLS 1.2 habilitado por padrão. Se você estiver executando o Windows 7 SP1 ou o Windows Server 2008 R2 SP1, precisará garantir que o suporte ao TLS 1.2 esteja habilitado no registro, conforme descrito em https://learn.microsoft.com/en-

us/windows- server/security/tls/tls -registry-settings #tls -12. Se estiver executando uma distribuição anterior, você deverá atualizar seu sistema operacional. Para obter informações sobre o suporte ao TLS 1.3 no Windows, consulte a documentação mais recente da Microsoft para ver as versões mínimas exigidas de cliente ou servidor.

macOS

Se estiver executando o .NET Core 2.1 ou posterior, o TLS 1.2 estará habilitado por padrão. O TLS 1.2 é compatível com o OS X Mavericks v10.9 ou posterior. .NET Core versão 2.1 e versões posteriores exigem versões mais recentes do macOS, conforme descrito em? https:// learn.microsoft.com/en-us/ dotnet/core/install/windows tabs=net80&pivots=os-macos.

Caso você esteja usando o .NET Core 1.0, o .NET Core usa OpenSSL no macOS, uma dependência que deve ser instalada separadamente. O OpenSSL adicionou suporte para TLS 1.2 na versão 1.0.1 e adicionou suporte para TLS 1.3 na versão 1.1.1.

Linux

O .NET Core no Linux requer OpenSSL, que é fornecido junto com muitas distribuições Linux. Mas também pode ser instalado separadamente. O OpenSSL adicionou suporte para TLS 1.2 na versão 1.0.1 e adicionou suporte para TLS 1.3 na versão 1.1.1. Caso você esteja usando uma versão moderna do .NET Core (2.1 ou posterior) e tenha instalado um gerenciador de pacotes, é provável que uma versão mais moderna do OpenSSL tenha sido instalada para você.

Para ter certeza, você pode executar o **`openssl version`** em um terminal e verificar se a versão é posterior à 1.0.1.

NET Framework

Se você estiver executando uma versão moderna do .NET Framework (4.7 ou posterior) e uma versão moderna do Windows (pelo menos o Windows 8 para clientes, o Windows Server 2012 ou posterior para servidores), o TLS 1.2 será habilitado e usado por padrão.

Se você estiver usando um runtime do.NET Framework que não usa as configurações do sistema operacional (.NET Framework 3.5 a 4.5.2), eles AWS SDK para .NET tentarão adicionar suporte para TLS 1.1 e TLS 1.2 aos protocolos suportados. Se você estiver usando o .NET Framework 3.5, esse processo só será bem-sucedido se o hot patch apropriado estiver instalado, da seguinte forma:

- Windows 10 versão 1511 e Windows Server 2016 — KB3156421
- Windows 8.1 e Windows Server 2012 R2 — KB3154520

- Windows Server 2012 — KB3154519
- Windows 7 SP1 e Server 2008 R2 SP1 — KB3154518

> **Warning**
>
> Em 15 de agosto de 2024, o suporte para o.NET Framework 3.5 AWS SDK para .NET foi encerrado e a versão mínima do.NET Framework foi alterada para 4.7.2. Para acessar mais informações, consulte a postagem no blog Important changes coming for .NET Framework 3.5 and 4.5 targets of the AWS SDK para .NET.

Se seu aplicativo estiver sendo executado em um novo.NET Framework no Windows 7 SP1 ou no Windows Server 2008 R2 SP1, você precisará garantir que o suporte ao TLS 1.2 esteja habilitado no registro, conforme descrito em https://learn.microsoft.com/en-us/windows- server/security/tls/tls -registry-settings #tls -12. Nas versões mais recentes do Windows, ele está habilitado por padrão.

Para obter as melhores práticas detalhadas para usar o TLS com o.NET Framework, consulte o artigo da Microsoft em https://learn.microsoft.com/en-us/dotnet/framework/network-programming/tls.

Ferramentas da AWS para PowerShell

Ferramentas da AWS para PowerShelluse o AWS SDK para .NET para todas as chamadas para AWS serviços. O comportamento do seu ambiente depende da versão do Windows PowerShell que você está executando, da seguinte forma.

Windows PowerShell 2.0 a 5.x

O Windows PowerShell 2.0 a 5.x é executado no.NET Framework. Você pode verificar qual runtime do.NET (2.0 ou 4.0) está sendo usado PowerShell usando o comando a seguir.

```
$PSVersionTable.CLRVersion
```

- Ao usar o .NET Runtime 2.0, siga as instruções fornecidas anteriormente em relação ao AWS SDK para .NET e ao .NET Framework 3.5.

> **Warning**
>
> Em 15 de agosto de 2024, o suporte para o.NET Framework 3.5 AWS SDK para .NET foi encerrado e a versão mínima do.NET Framework foi alterada para 4.7.2. Para acessar

mais informações, consulte a postagem no blog Important changes coming for .NET Framework 3.5 and 4.5 targets of the AWS SDK para .NET.

- Ao usar o .NET Runtime 4.0, siga as instruções fornecidas anteriormente em relação ao AWS SDK para .NET e ao .NET Framework 4+.

Windows PowerShell 6.0

O Windows PowerShell 6.0 e versões mais recentes são executados no.NET Core. Você pode verificar qual versão do .NET Core está sendo usada executando o comando a seguir.

```
[System.Reflection.Assembly]::GetEntryAssembly().GetCustomAttributes([System.Runtime.Versioning
 $true).FrameworkName
```

Siga as instruções fornecidas anteriormente em relação à versão relevante do.NET Core AWS SDK para .NET e à respectiva.

Xamarin

Para o Xamarin, veja as instruções em -layer-security. https://learn.microsoft.com/en-us/ xamarin/ cross-platform/app-fundamentals/transport Em resumo:

Para Android

- Requer o Android 5.0 ou posterior.
- Propriedades do projeto, opções do Android: a HttpClient implementação deve ser definida como Android e a SSL/TLS implementação definida como TLS 1.2+ nativo.

Para iOS

- Requer o iOS 7 ou posterior.
- Propriedades do projeto, versão iOS: HttpClient a implementação deve ser definida como NSUrlSessão.

No macOS

- Requer o macOS 10.9 ou posterior.

- Opções do projeto, compilação, compilação do Mac: HttpClient a implementação deve ser definida como NSUrlSessão.

Unity

Você deve usar o Unity 2018.2 ou posterior e usar o tempo de execução de scripts equivalente ao .NET 4.x. Você pode definir isso em Configurações do projeto, Configuração, Player, conforme descrito em https://docs.unity3d.com/2019.1/Documentation/Manual/ScriptingRuntimeUpgrade.html. O tempo de execução de script equivalente do.NET 4.x permite o suporte ao TLS 1.2 em todas as plataformas Unity que executam Mono ou CPP. IL2

Navegador (para Blazor) WebAssembly

WebAssembly é executado no navegador em vez de no servidor e usa o navegador para lidar com o tráfego HTTP. Portanto, o suporte a TLS é determinado pelo suporte do navegador.

O Blazor WebAssembly, na versão prévia do ASP.NET Core 3.1, é suportado somente em navegadores que oferecem suporte WebAssembly, conforme descrito em -platforms. https://learn.microsoft.com/en-us/ aspnet/core/blazor/supported Todos os principais navegadores eram compatíveis com o TLS 1.2 antes do suporte. WebAssembly Se esse for o caso do seu navegador, se o seu aplicativo for executado, ele poderá se comunicar por TLS 1.2.

Consulte a documentação do seu navegador para obter mais informações e verificação.

Migração do cliente de criptografia S3 (V1 para V2)

> **Note**
>
> Se você estiver usando a V2 e quiser migrar para a V4, consulte. Migração do cliente de criptografia S3 (V2 para V4)

Este tópico mostra como migrar seus aplicativos da versão 1 (V1) do cliente de criptografia Amazon Simple Storage Service (S3) para a versão 2 (V2) e garantir a disponibilidade dos aplicativos durante todo o processo de migração.

Objetos criptografados com o cliente V2 não podem ser descriptografados com o cliente V1. Para facilitar a migração para o novo cliente sem precisar recriptografar todos os objetos de uma vez,

foi fornecido um cliente “V1 de transição”. Esse cliente pode descriptografar objetos criptografados com a V1 e V2, mas criptografa somente objetos em formato compatível com a V1. O cliente V2 pode descriptografar objetos criptografados em V1 e V2 (quando habilitado para objetos V1), mas criptografa objetos somente em formato compatível com V2.

Visão geral da migração

Essa migração acontece em três fases: Essas fases são apresentadas aqui e serão descritas em detalhes posteriormente. Cada fase deve ser concluída para todos os clientes que usam objetos compartilhados antes que a próxima fase seja iniciada.

1. Atualize os clientes existentes para clientes V1 de transição para ler novos formatos. Primeiro, atualize seus aplicativos para que dependam do cliente V1 de transição em vez do cliente V1. O cliente V1 de transição permite que seu código existente descriptografe objetos escritos pelos novos clientes V2 e objetos escritos em formato compatível com a V1.

   > **Note**
   >
   > O cliente V1 de transição é fornecido somente para fins de migração. Prossiga com a atualização para o cliente V2 depois de passar para o cliente de transição V1.

2. Migre clientes V1 de transição para clientes V2 para escrever novos formatos. Em seguida, substitua todos os clientes V1 de transição em seus aplicativos por clientes V2 e defina o perfil de segurança como `V2AndLegacy`. Definir esse perfil de segurança em clientes V2 permite que esses clientes descriptografem objetos que foram criptografados em formato compatível com V1.
3. Atualize os clientes V2 para que não leiam mais os formatos V1. Por fim, depois que todos os clientes tiverem migrado para a V2 e todos os objetos tiverem sido criptografados ou recriptografados em um formato compatível com a V2, defina o perfil de segurança da V2 como `V2` em vez de `V2AndLegacy`. Isso evita a descriptografia de objetos que estão no formato compatível com V1.

Atualize os clientes existentes para clientes V1 de transição para ler novos formatos.

O cliente de criptografia V2 usa algoritmos de criptografia incompatíveis com as versões mais antigas do cliente. A primeira etapa da migração é atualizar seus clientes de descriptografia V1 para que eles possam ler o novo formato.

O cliente V1 de transição permite que seus aplicativos descriptografem objetos criptografados na V1 e V2. Esse cliente faz parte do pacote NuGet Amazon.Extensions.S3.Encryption. Execute as etapas a seguir em cada um dos seus aplicativos para usar o cliente V1 de transição.

1. Adquira uma nova dependência do pacote Amazon.Extensions.S3.Encryption. Se o seu projeto depende diretamente do AWSSDK.S3 ou. AWSSDK KeyManagementServicepacotes, você deve atualizar essas dependências ou removê-las para que suas versões atualizadas sejam inseridas com esse novo pacote.
2. Altere a declaração `using` apropriada de `Amazon.S3.Encryption` para `Amazon.Extensions.S3.Encryption`, da seguinte forma:

```
// using Amazon.S3.Encryption;
  using Amazon.Extensions.S3.Encryption;
```

3. Reconstrua e reimplante o aplicativo.

O cliente V1 de transição é totalmente compatível com a API do cliente V1, portanto, nenhuma outra alteração de código é necessária.

Migre clientes V1 de transição para clientes V2 para escrever novos formatos.

O cliente V2 faz parte do pacote NuGet Amazon.Extensions.S3.Encryption. Ele permite que seus aplicativos descriptografem objetos criptografados na V1 e V2 (se configurados para fazer isso), mas criptografa objetos somente em formato compatível com V2.

Depois de atualizar seus clientes existentes para ler o novo formato de criptografia, você poderá prosseguir com a atualização segura de seus aplicativos para os clientes de criptografia e descriptografia V2. Execute as etapas a seguir em cada um dos seus aplicativos para usar o cliente V2.

1. Fazer `EncryptionMaterials` alteração `EncryptionMaterialsV2`.
    a. Ao usar o KMS:
        i. Forneça um ID da chave do KMS.
        ii. Declare o método de criptografia que você está usando; ou seja, `KmsType.KmsContext`.

iii. Forneça um contexto de criptografia ao KMS para associar a essa chave de dados. Você pode enviar um dicionário vazio (o contexto de criptografia da Amazon ainda será mesclado), mas é recomendável fornecer contexto adicional.

b. Ao usar métodos de wrapping da chave fornecidos pelo usuário (criptografia simétrica ou assimétrica):

i. Forneça uma instância `AES` ou `RSA` que contenha os materiais de criptografia.

ii. Declare qual algoritmo de criptografia usar; ou seja, `SymmetricAlgorithmType.AesGcm` ou `AsymmetricAlgorithmType.RsaOaepSha1`.

2. Altere `AmazonS3CryptoConfiguration` para `AmazonS3CryptoConfigurationV2` com a propriedade `SecurityProfile` definida como `SecurityProfile.V2AndLegacy`.
3. Fazer `AmazonS3EncryptionClient` alteração `AmazonS3EncryptionClientV2`. Esse cliente pega os objetos `AmazonS3CryptoConfigurationV2` e `EncryptionMaterialsV2` recém-convertidos das etapas anteriores.

Exemplo: KMS para KMS+Context

Pré-migração

```
using System.Security.Cryptography;
using Amazon.S3.Encryption;

var encryptionMaterial = new
 EncryptionMaterials("1234abcd-12ab-34cd-56ef-1234567890ab");
var configuration = new AmazonS3CryptoConfiguration()
{
    StorageMode = CryptoStorageMode.ObjectMetadata
};
var encryptionClient = new AmazonS3EncryptionClient(configuration, encryptionMaterial);
```

Pós-migração

```
using System.Security.Cryptography;
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;

var encryptionContext = new Dictionary<string, string>();
```

```
var encryptionMaterial = new
 EncryptionMaterialsV2("1234abcd-12ab-34cd-56ef-1234567890ab", KmsType.KmsContext,
 encryptionContext);
var configuration = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2AndLegacy)
{
    StorageMode = CryptoStorageMode.ObjectMetadata
};
var encryptionClient = new AmazonS3EncryptionClientV2(configuration,
 encryptionMaterial);
```

Exemplo: Algoritmo simétrico (AES-CBC para AES-GCM Key Wrap)

`StorageMode` pode ser `ObjectMetadata` ou `InstructionFile`.

Pré-migração

```
using System.Security.Cryptography;
using Amazon.S3.Encryption;

var symmetricAlgorithm = Aes.Create();
var encryptionMaterial = new EncryptionMaterials(symmetricAlgorithm);
var configuration = new AmazonS3CryptoConfiguration()
{
    StorageMode = CryptoStorageMode.ObjectMetadata
};
var encryptionClient = new AmazonS3EncryptionClient(configuration, encryptionMaterial);
```

Pós-migração

```
using System.Security.Cryptography;
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;

var symmetricAlgorithm = Aes.Create();
var encryptionMaterial = new EncryptionMaterialsV2(symmetricAlgorithm,
 SymmetricAlgorithmType.AesGcm);
var configuration = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2AndLegacy)
{
    StorageMode = CryptoStorageMode.ObjectMetadata
};
var encryptionClient = new AmazonS3EncryptionClientV2(configuration,
 encryptionMaterial);
```

> **Note**
>
> Ao descriptografar com o AES-GCM, leia o objeto inteiro até o fim antes de começar a usar os dados descriptografados. Isso é para verificar se o objeto não foi modificado desde que foi criptografado.

Exemplo: Algoritmo assimétrico (RSA para RSA-OAEP-SHA 1 quebra de chave)

`StorageMode` pode ser `ObjectMetadata` ou `InstructionFile`.

Pré-migração

```
using System.Security.Cryptography;
using Amazon.S3.Encryption;

var asymmetricAlgorithm = RSA.Create();
var encryptionMaterial = new EncryptionMaterials(asymmetricAlgorithm);
var configuration = new AmazonS3CryptoConfiguration()
{
    StorageMode = CryptoStorageMode.ObjectMetadata
};
var encryptionClient = new AmazonS3EncryptionClient(configuration, encryptionMaterial);
```

Pós-migração

```
using System.Security.Cryptography;
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;

var asymmetricAlgorithm = RSA.Create();
var encryptionMaterial = new EncryptionMaterialsV2(asymmetricAlgorithm,
 AsymmetricAlgorithmType.RsaOaepSha1);
var configuration = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2AndLegacy)
{
    StorageMode = CryptoStorageMode.ObjectMetadata
};
var encryptionClient = new AmazonS3EncryptionClientV2(configuration,
 encryptionMaterial);
```

Atualize os clientes V2 para que não leiam mais os formatos V1.

Eventualmente, todos os objetos terão sido criptografados ou recriptografados usando um cliente V2. Depois que essa conversão for concluída, você poderá desativar a compatibilidade com a V1 nos clientes V2 definindo a propriedade `SecurityProfile` como `SecurityProfile.V2`, conforme mostrado no trecho a seguir.

```
//var configuration = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2AndLegacy);
var configuration = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2);
```

Migração do cliente de criptografia S3 (V2 para V4)

> **Note**
>
> Se você estiver usando a V1 e quiser migrar para a V4, primeiro deverá migrar para a V2. Consulte Migração do cliente de criptografia S3 (V1 para V2).

Este tópico mostra como migrar seus aplicativos da versão 2 (V2) do cliente de criptografia Amazon Simple Storage Service (S3) para a versão 4 (V4) e garantir a disponibilidade dos aplicativos durante todo o processo de migração. A V4 usa o AES-GCM com o compromisso principal de criptografia de conteúdo e apresenta políticas de compromisso para aumentar a segurança contra ataques de substituição de chaves.

O cliente V4 está disponível no pacote NuGet Amazon.Extensions.S3.Encryption.

> **Important**
>
> Alterações importantes: clientes V4 configurados com `RequireEncryptRequireDecrypt` política não podem descriptografar objetos criptografados com clientes V1 ou V2. Somente os clientes V2 mais recentes podem descriptografar objetos criptografados em V4 com comprometimento de chave. Antes de passar para a `RequireEncryptRequireDecrypt` política, você deve criptografar novamente todos os seus dados existentes usando clientes V4 com o compromisso de chave ativado.

Compreendendo os conceitos da V4

A V4 usa o AES-GCM com o compromisso fundamental de criptografia de conteúdo e apresenta os principais conceitos de segurança que aprimoram a proteção de seus dados criptografados:

Política de compromisso

A Política de Compromisso controla como o cliente de criptografia lida com o comprometimento da chave durante as operações de criptografia e descriptografia. A V4 oferece suporte a três políticas de compromisso:

`ForbidEncryptAllowDecrypt`

- Criptografia: sem compromisso
- Descriptografia: permite que objetos não confirmem
- Segurança: não impõe o compromisso e pode permitir a adulteração
- Compatibilidade: todas as implementações V2 e V4 podem ler objetos criptografados com esta política

`RequireEncryptAllowDecrypt`

- Criptografia: com compromisso fundamental
- Descriptografia: permite que objetos confirmados e não confirmados
- Segurança: novos objetos são protegidos contra ataques de substituição de chaves, objetos antigos ainda podem ser lidos
- Compatibilidade: somente a V4 oferece suporte a essa política

`RequireEncryptRequireDecrypt`(Padrão para V4)

- Criptografia: com compromisso fundamental
- Descriptografia: somente confirmando objetos
- Segurança: cumprimento total do compromisso para máxima segurança
- Compatibilidade: somente a V4 oferece suporte a essa política

AES GCM com compromisso fundamental

A V4 usa o AES-GCM com o compromisso fundamental de criptografia de conteúdo, o que fornece segurança aprimorada:

- Proteção contra adulteração: protege contra ataques de substituição de chave vinculando criptograficamente a chave aos dados criptografados.
- Compatibilidade de versão: objetos criptografados com comprometimento de chave só podem ser descriptografados por clientes V4 e versões mais recentes.

> **Warning**
>
> Antes de ativar a criptografia de comprometimento de chave na produção, certifique-se de que todos os aplicativos que precisam descriptografar seus objetos tenham sido atualizados para a V4 ou mais recente, pois o cliente V2 está se tornando obsoleto.

Atualize os clientes existentes para ler os formatos V4

O cliente de criptografia V4 usa algoritmos de criptografia que as versões mais antigas do cliente não suportam. A primeira etapa da migração é atualizar seus clientes V2 para que eles possam ler o novo formato V4.

Atualizar dependências NuGet do Package

Atualize seus aplicativos para usar a versão mais recente do pacote NuGet Amazon.Extensions.S3.Encryption que inclui suporte à V4. Execute as seguintes etapas em cada um dos seus aplicativos:

1. Atualize para o pacote Amazon.Extensions.S3.Encryption mais recente. Se o seu projeto depende diretamente do AWSSDK.S3 ou. AWSSDK KeyManagementServicepacotes, você deve atualizar essas dependências ou removê-las para que suas versões atualizadas sejam inseridas com esse novo pacote.
2. Certifique-se de que suas `using` declarações façam referência ao namespace correto:

```
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;
```

3. Reconstrua e reimplante o aplicativo.

Seus clientes V2 existentes continuarão trabalhando com o pacote atualizado e poderão descriptografar objetos criptografados por clientes V4 (dependendo da política de compromisso usada).

Crie e implante aplicativos

Depois de atualizar as dependências NuGet do pacote:

1. Crie seu aplicativo para garantir que todas as dependências sejam resolvidas corretamente.
2. Teste seu aplicativo em um ambiente de desenvolvimento para verificar se a funcionalidade existente continua funcionando.
3. Implante o aplicativo atualizado em seu ambiente de produção.

Essa atualização permite que seus clientes V2 existentes descriptografem objetos que serão criptografados por clientes V4, garantindo compatibilidade durante o processo de migração.

Migre para clientes V4

Depois de atualizar seus clientes existentes para ler o novo formato de criptografia, você pode continuar atualizando seus aplicativos com segurança para usar clientes de criptografia e descriptografia V4. O cliente V4 fornece segurança aprimorada por meio do comprometimento de chaves, mantendo a compatibilidade com objetos criptografados existentes.

Processo de migração em 4 etapas

A migração da V2 para a V4 segue um processo estruturado de 4 etapas para garantir compatibilidade e segurança. Cada etapa representa uma configuração específica que deve ser implantada em todos os seus aplicativos antes de prosseguir para a próxima etapa.

1. Etapa 0: Cliente V2 (ponto de partida) - Sua implementação V2 existente
2. Etapa 1: V4 com compatibilidade V2 - Migre para o cliente V4 enquanto mantém o comportamento de criptografia compatível com V2
3. Etapa 2: V4 com gravações de compromisso de chave - Comece a criptografar com compromisso de chave enquanto permite a decodificação de objetos legados
4. Etapa 3: V4 com aplicação total - Exigir comprometimento de chave para criptografia e decodificação

Etapa 0: Cliente V2 (ponto de partida)

Isso representa sua configuração de cliente V2 existente. Essa etapa demonstra o estado inicial antes da migração.

```
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;
using Amazon.S3.Model;

// Step 0: V2 Client - Starting configuration
var encryptionContext = new Dictionary<string, string>();
var encryptionMaterial = new EncryptionMaterialsV2(kmsKeyId, KmsType.KmsContext,
 encryptionContext);

#pragma warning disable 0618
var configuration = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2);
#pragma warning enable 0618

var encryptionClient = new AmazonS3EncryptionClientV2(configuration,
 encryptionMaterial);

// Use the client for PutObject and GetObject operations
await encryptionClient.PutObjectAsync(new PutObjectRequest
{
    BucketName = bucket,
    Key = objectKey,
    ContentBody = content
});
```

Etapa 1: V4 com compatibilidade com V2

Migre para o cliente V4 enquanto mantém um comportamento idêntico ao V2. Essa etapa usa a `ForbidEncryptAllowDecrypt` política para criptografar sem compromisso e permitir a descriptografia de todos os objetos.

```
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;
using Amazon.S3.Model;

// Step 1: V4 Client with V2 compatibility
var encryptionContext = new Dictionary<string, string>();
```

```
var encryptionMaterial = new EncryptionMaterialsV4(kmsKeyId, KmsType.KmsContext,
 encryptionContext);

var configuration = new AmazonS3CryptoConfigurationV4(
    SecurityProfile.V4,
    CommitmentPolicy.ForbidEncryptAllowDecrypt,
    ContentEncryptionAlgorithm.AesGcm);

var encryptionClient = new AmazonS3EncryptionClientV4(configuration,
 encryptionMaterial);

// Use the client for PutObject and GetObject operations
await encryptionClient.PutObjectAsync(new PutObjectRequest
{
    BucketName = bucket,
    Key = objectKey,
    ContentBody = content
});
```

Comportamento: criptografa sem compromisso, pode descriptografar objetos comprometidos e não comprometidos. Idêntico ao comportamento da V2.

Etapa 2: V4 com gravações de compromisso importantes

Comece a criptografar com o compromisso de chave e, ao mesmo tempo, mantenha a compatibilidade com versões anteriores para decodificação. Essa etapa usa a `RequireEncryptAllowDecrypt` política.

> **Warning**
>
> Antes de implantar a Etapa 2, certifique-se de que todos os leitores tenham sido atualizados para a Etapa 1 ou posterior para lidar com a criptografia de compromisso de chave.

```
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;
using Amazon.S3.Model;

// Step 2: V4 Client with key commitment writes
var encryptionContext = new Dictionary<string, string>();
var encryptionMaterial = new EncryptionMaterialsV4(kmsKeyId, KmsType.KmsContext,
 encryptionContext);
```

```
var configuration = new AmazonS3CryptoConfigurationV4(
    SecurityProfile.V4,
    CommitmentPolicy.RequireEncryptAllowDecrypt,
    ContentEncryptionAlgorithm.AesGcmWithCommitment);

var encryptionClient = new AmazonS3EncryptionClientV4(configuration,
 encryptionMaterial);

// Use the client for PutObject and GetObject operations
await encryptionClient.PutObjectAsync(new PutObjectRequest
{
    BucketName = bucket,
    Key = objectKey,
    ContentBody = content
});
```

Comportamento: criptografa com comprometimento, pode descriptografar objetos comprometidos e não comprometidos. Novos objetos são protegidos contra ataques de substituição de chaves.

Etapa 3: V4 com aplicação total

Exija o comprometimento da chave tanto para criptografia quanto para decodificação. Essa etapa usa a `RequireEncryptRequireDecrypt` política para máxima segurança.

> **Warning**
>
> Antes de implantar a Etapa 3, certifique-se de que todos os objetos em seu sistema tenham sido criptografados novamente com o comprometimento da chave (Etapa 2). Essa etapa falhará ao descriptografar objetos criptografados sem compromisso.

```
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;
using Amazon.S3.Model;

// Step 3: V4 Client with full key commitment enforcement
var encryptionContext = new Dictionary<string, string>();
var encryptionMaterial = new EncryptionMaterialsV4(kmsKeyId, KmsType.KmsContext,
 encryptionContext);
```

```
var configuration = new AmazonS3CryptoConfigurationV4(
    SecurityProfile.V4,
    CommitmentPolicy.RequireEncryptRequireDecrypt,
    ContentEncryptionAlgorithm.AesGcmWithCommitment);

var encryptionClient = new AmazonS3EncryptionClientV4(configuration,
 encryptionMaterial);

// Use the client for PutObject and GetObject operations
await encryptionClient.PutObjectAsync(new PutObjectRequest
{
    BucketName = bucket,
    Key = objectKey,
    ContentBody = content
});
```

Comportamento: criptografa com compromisso, somente descriptografa objetos criptografados com compromisso. Segurança máxima contra ataques de substituição de chaves.

Exemplos adicionais de configuração

Esta seção fornece exemplos adicionais para configurar clientes V4 com opções diferentes durante a migração.

Habilitando o Legacy Support

Para permitir que clientes V4 leiam objetos criptografados por clientes V1 e V2, configure o cliente com uma política de compromisso que permita a descriptografia legada:

```
using Amazon.Extensions.S3.Encryption;
using Amazon.Extensions.S3.Encryption.Primitives;

// Configure V4 client to read V1/V2 objects
var configuration = new AmazonS3CryptoConfigurationV4(SecurityProfile.V4AndLegacy,
 CommitmentPolicy.RequireEncryptAllowDecrypt)
{
    StorageMode = CryptoStorageMode.ObjectMetadata
};

// This configuration allows:
// - Encryption: With commitment (secure)
// - Decryption: Both V2 (non-committing) and V4 (committing) objects
```

Use essa configuração durante a migração quando precisar descriptografar objetos criptografados por clientes mais antigos e, ao mesmo tempo, garantir que novos objetos sejam criptografados com segurança aprimorada.

Configurando o método de armazenamento

A V4 oferece suporte a dois métodos de armazenamento para metadados de criptografia. Escolha o método mais adequado ao seu caso de uso:

Metadados do objeto (padrão)

```
var configuration = new AmazonS3CryptoConfigurationV4(SecurityProfile.V4,
 CommitmentPolicy.RequireEncryptAllowDecrypt)
{
    StorageMode = CryptoStorageMode.ObjectMetadata
};
// Encryption metadata is stored in S3 object metadata
```

Arquivo de instruções

```
var configuration = new AmazonS3CryptoConfigurationV4(SecurityProfile.V4,
 CommitmentPolicy.RequireEncryptAllowDecrypt)
{
    StorageMode = CryptoStorageMode.InstructionFile
};
// Encryption metadata is stored in a separate S3 object (instruction file)
```

Use `InstructionFile` quando precisar preservar os metadados do objeto para outras finalidades ou ao trabalhar com objetos que tenham limitações de tamanho de metadados.

Configurando a política de compromisso

Escolha a política de compromisso apropriada com base em seus requisitos de segurança e fase de migração:

Fase de migração (compatibilidade V2)

```
// For migration: encrypt without commitment, allow all decryption
var migrationConfig = new AmazonS3CryptoConfigurationV4(SecurityProfile.V4,
 CommitmentPolicy.ForbidEncryptAllowDecrypt);
```

Fase de transição (recomendada)

```
// For transition: encrypt with commitment, allow legacy decryption
var transitionConfig = new AmazonS3CryptoConfigurationV4(SecurityProfile.V4,
 CommitmentPolicy.RequireEncryptAllowDecrypt);
```

Fase de segurança completa

```
// For maximum security: require commitment for both encryption and decryption
var secureConfig = new AmazonS3CryptoConfigurationV4(SecurityProfile.V4,
 CommitmentPolicy.RequireEncryptRequireDecrypt);
```

Comece `ForbidEncryptAllowDecrypt` durante a migração inicial, passe `RequireEncryptAllowDecrypt` para a fase de transição e, finalmente, use `RequireEncryptRequireDecrypt` quando todos os clientes tiverem sido atualizados e todos os objetos tiverem sido criptografados novamente com compromisso.

Considerações especiais para o AWS SDK para .NET

Esta seção contém considerações para casos especiais em que as configurações ou procedimentos normais não são adequados ou suficientes.

Tópicos

- Obtenção de montagens para o AWS SDK para .NET
- Acessar credenciais e perfis em um aplicativo
- Considerações especiais sobre o suporte ao Unity
- Considerações especiais sobre o suporte do Xamarin

Obtenção de montagens para o AWS SDK para .NET

Este tópico descreve como você pode obter as AWSSDK montagens e armazená-las localmente (ou localmente) para uso em seus projetos. Esse não é o método recomendado para lidar com referências do SDK, mas é obrigatório em alguns ambientes.

> **Note**
>
> O método recomendado para lidar com referências do SDK é baixar e instalar apenas os NuGet pacotes de que cada projeto precisa. Esse método é descrito em Instale AWSSDK pacotes com NuGet.

Se você não puder ou não tiver permissão para baixar e instalar NuGet pacotes por projeto, as seguintes opções estão disponíveis para você.

Faça download e extraia os arquivos ZIP

(Lembre-se de que esse não é o método recomendado para lidar com referências ao AWS SDK para .NET.)

1. Faça download de um dos arquivos ZIP a seguir:
   - aws-sdk-net8.0.zip - Assemblies que suportam o.NET 8 e versões posteriores.

- aws-sdk-netcoreapp3.1.zip - Assemblies que suportam o.NET Core 3.1 e versões posteriores.
- aws-sdk-netstandard2.0.zip - Assemblies que suportam o.NET Standard 2.0 e 2.1.
- aws-sdk-net472.zip - Assemblies que suportam o.NET Framework 4.5 e versões posteriores.

2. Extraia os assemblies em alguma pasta de “download” em seu sistema de arquivos; não importa onde. Anote qual é essa pasta.
3. Ao configurar seu projeto, você obterá os assemblies necessários a partir dessa pasta, conforme descrito em Instale AWSSDK montagens sem NuGet.

Acessar credenciais e perfis em um aplicativo

O método preferido para usar credenciais é permitir que o as AWS SDK para .NET encontre e recupere para você, conforme descrito em. Resolução de perfil e credenciais

No entanto, você também pode configurar seu aplicativo para recuperar ativamente perfis e credenciais e, em seguida, usar essas credenciais explicitamente ao criar um cliente de serviço. AWS

Para recuperar ativamente perfis e credenciais, use classes do Amazon.Runtime. CredentialManagementnamespace.

- Para encontrar um perfil em um arquivo que usa o formato de arquivo de AWS credenciais (o arquivo de AWS credenciais compartilhado em seu local padrão ou um arquivo de credenciais personalizado), use a classe. SharedCredentialsFile Para resumir, algumas vezes, arquivos nesse formato são chamados simplesmente de arquivos de credenciais neste texto.
- Para encontrar um perfil no SDK Store, use a classe Net SDKCredentials File.
- Para pesquisar em um arquivo de credenciais e no SDK Store, dependendo da configuração de uma propriedade de classe, use a CredentialProfileStoreChainclasse.

  Você pode usar essa classe para encontrar perfis. Você também pode usar essa classe para solicitar AWS credenciais diretamente em vez de usar a `AWSCredentialsFactory` classe (descrita a seguir).
- Para recuperar ou criar vários tipos de credenciais de um perfil, use a classe AWSCredentialsFactory.

As seções a seguir fornecem exemplos dessas classes.

Exemplos para a aula CredentialProfileStoreChain

Você pode obter credenciais ou perfis da CredentialProfileStoreChainturma usando os TryGetProfilemétodos TryGetAWSCredentialsou. A propriedade `ProfilesLocation` da classe determina o comportamento dos métodos, da seguinte forma:

- Se `ProfilesLocation` for nulo ou vazio, pesquise no SDK Store se a plataforma for compatível e, em seguida, pesquise o arquivo de AWS credenciais compartilhado no local padrão.
- Se a propriedade `ProfilesLocation` contiver um valor, pesquise o arquivo de credenciais especificado na propriedade.

Obtenha credenciais do SDK Store ou do arquivo de credenciais compartilhado AWS

Este exemplo mostra como obter credenciais usando a classe `CredentialProfileStoreChain` e, em seguida, use as credenciais para criar um objeto AmazonS3Client. As credenciais podem vir da SDK Store ou do arquivo de credenciais compartilhado da AWS no local padrão.

Este exemplo também usa o Amazon.Runtime. AWSCredentialsclasse.

```
var chain = new CredentialProfileStoreChain();
AWSCredentials awsCredentials;
if (chain.TryGetAWSCredentials("some_profile", out awsCredentials))
{
    // Use awsCredentials to create an Amazon S3 service client
    using (var client = new AmazonS3Client(awsCredentials))
    {
        var response = await client.ListBucketsAsync();
        Console.WriteLine($"Number of buckets: {response.Buckets.Count}");
    }
}
```

Obtenha um perfil na SDK Store ou no arquivo de AWS credenciais compartilhado

Este exemplo mostra como obter um perfil usando a CredentialProfileStoreChain classe. As credenciais podem vir do SDK Store ou do arquivo de AWS credenciais compartilhado no local padrão.

Este exemplo também usa a CredentialProfileclasse.

```
var chain = new CredentialProfileStoreChain();
```

```
CredentialProfile basicProfile;
if (chain.TryGetProfile("basic_profile", out basicProfile))
{
    // Use basicProfile
}
```

Obtenha credenciais de um arquivo de credenciais personalizado

Este exemplo mostra como obter credenciais usando a CredentialProfileStoreChain classe. As credenciais vêm de um arquivo que usa o formato de arquivo de AWS credenciais, mas está em um local alternativo.

Este exemplo também usa o Amazon.Runtime. AWSCredentialsclasse.

```
var chain = new
    CredentialProfileStoreChain("c:\\Users\\sdkuser\\customCredentialsFile.ini");
AWSCredentials awsCredentials;
if (chain.TryGetAWSCredentials("basic_profile", out awsCredentials))
{
    // Use awsCredentials to create an AWS service client
}
```

Exemplos de aulas SharedCredentialsFile e AWSCredentials fábrica

Crie um AmazonS3Client usando a classe SharedCredentialsFile

Este exemplo mostra como encontrar um perfil no arquivo de AWS credenciais compartilhado, criar credenciais a partir do perfil e, em seguida, usar AWS as credenciais para criar um objeto AmazonS3Client. O exemplo usa a SharedCredentialsFileclasse.

Este exemplo também usa a CredentialProfileclasse e o Amazon.Runtime. AWSCredentialsclasse.

```
CredentialProfile basicProfile;
AWSCredentials awsCredentials;
var sharedFile = new SharedCredentialsFile();
if (sharedFile.TryGetProfile("basic_profile", out basicProfile) &&
    AWSCredentialsFactory.TryGetAWSCredentials(basicProfile, sharedFile, out
 awsCredentials))
{
    // use awsCredentials to create an Amazon S3 service client
    using (var client = new AmazonS3Client(awsCredentials, basicProfile.Region))
```

```
    {
        var response = await client.ListBucketsAsync();
        Console.WriteLine($"Number of buckets: {response.Buckets.Count}");
    }
}
```

> **Note**
>
> A classe Net SDKCredentials File pode ser usada exatamente da mesma forma, exceto que você instanciaria um novo objeto Net SDKCredentials File em vez de um SharedCredentialsFile objeto.

Considerações especiais sobre o suporte ao Unity

Ao usar o.NET Standard 2.0 para seu aplicativo Unity, seu aplicativo deve referenciar os AWS SDK para .NET assemblies (arquivos DLL) diretamente em vez de usar. AWS SDK para .NET NuGet Dado esse requisito, a seguir estão as ações importantes que você precisará realizar.

- Você precisa obter os AWS SDK para .NET conjuntos e aplicá-los ao seu projeto. Para obter informações sobre como fazer isso, consulte Faça download e extraia os arquivos ZIP no tópico Obtenção de AWSSDK montagens.

- Você precisa incluir o seguinte DLLs em seu projeto Unity junto com o DLLs for AWSSDK.Core e os outros AWS serviços que você está usando. A partir da versão 3.5.109 do AWS SDK para .NET, o arquivo ZIP padrão.NET contém esses adicionais. DLLs
  - Microsoft.Bcl. AsyncInterfaces.dll
  - System.Runtime. CompilerServices.Inseguro.dll
  - System.Threading.Tasks.Extensions.dll

- Se você estiver usando o IL2CPP para criar seu projeto Unity, deverá adicionar um `link.xml` arquivo à sua pasta Asset para evitar a remoção do código. O `link.xml` arquivo deve listar todos os AWSSDK assemblies que você está usando e cada um deve incluir o `preserve="all"` atributo. O trecho a seguir mostra um exemplo desse arquivo.

```
<linker>
```

```
    <assembly fullname="AWSSDK.Core" preserve="all"/>
    <assembly fullname="AWSSDK.DynamoDBv2" preserve="all"/>
    <assembly fullname="AWSSDK.Lambda" preserve="all"/>
</linker>
```

> **Note**
>
> Para ler informações básicas interessantes relacionadas a esse requisito, consulte o artigo em https://aws.amazon.com/blogs/developer/ referencing-the-aws-sdk - for-net-standard -2-0- -uwp/. from-unity-xamarin-or

Considerações especiais sobre o suporte do Xamarin

Os projetos Xamarin (novos e existentes) devem ter como destino o .NET Standard 2.0. Consulte Suporte do .NET Standard 2.0 no Xamarin.Forms e Suporte à implementação do .NET.

Veja também as informações sobre a Portable Class Library e o Xamarin.

Referência de API para o AWS SDK para .NET

O AWS SDK para .NET fornece uma API que você pode usar para acessar AWS serviços. Para ver quais classes e métodos estão disponíveis na API, consulte a Referência de API do AWS SDK para .NET.

Além da referência geral fornecida acima, cada um dos exemplos na seção Exemplos de código guiado contém referências aos métodos e classes específicos que são usados nesse exemplo.

Sobre as versões de referência da API

A referência da API descrita anteriormente é para a versão 3.0 e posterior do AWS SDK para .NET.

Para obter informações sobre a migração de versões mais antigas do SDK, consulte Migrando seu projeto

Para encontrar conteúdo obsoleto para versões anteriores da referência da SDK API, consulte o (s) seguinte (s) item (s):

- AWS SDK para .NET Referência de API V1 (obsoleta)
- AWS SDK para .NET Referência de API V2 (descontinuada)

Para visualizar uma referência de AWS SDK para .NET API obsoleta, você precisará extraí-la e configurar um navegador da web. As instruções mostradas a seguir são exemplos de como fazer isso. Eles são baseados no uso do navegador Google Chrome em um sistema Windows. Adapte-os ao seu navegador e sistema operacional específicos.

Veja a referência de API obsoleta V1

1. Faça o download do arquivo ZIP da referência de AWS SDK para .NET API obsoleta V1. Por padrão, o nome do arquivo ZIP é `sdkfornet-api-ref_v1_deprecated.zip`. Esse nome será usado em todas essas instruções.
2. Coloque o arquivo ZIP em uma pasta de sua escolha. Para essas instruções, presume-se que o nome da pasta seja `C:\work\temp\api-refs\V1`.

3. Clique com o botão direito do mouse no arquivo ZIP e escolha Extrair tudo. Aceite o local padrão, que é `C:\work\temp\api-refs\V1\sdkfornet-api-ref_v1` para essas instruções.
4. Crie um atalho para o Google Chrome na `C:\work\temp\api-refs\V1` pasta. Tenha cuidado para não mover o aplicativo original.
5. Nas propriedades do novo atalho, defina os seguintes campos:

   - Alvo: `"<path to the Google Chrome application>" --disable-web-security --user-data-dir=C:\work\temp\api-refs\V1\data "C:\work\temp\api-refs\V1\sdkfornet-api-ref_v1\Index.html"`

     Para o `--user-data-dir` argumento, use um nome de pasta que funcione para o seu ambiente. A pasta não precisa existir.
   - Comece em: `C:\work\temp\api-refs\V1`
6. Dê ao atalho um nome apropriado.
7. Abra o atalho para ver a referência antiga da API.

Veja a referência de API obsoleta V2

1. Faça o download do arquivo ZIP da referência de AWS SDK para .NET API obsoleta V2. Por padrão, o nome do arquivo ZIP é`sdkfornet-api-ref_v2_deprecated.zip`. Esse nome será usado em todas essas instruções.
2. Coloque o arquivo ZIP em uma pasta de sua escolha. Para essas instruções, presume-se que o nome da pasta seja`C:\work\temp\api-refs\V2`.
3. Clique com o botão direito do mouse no arquivo ZIP e escolha Extrair tudo. Aceite o local padrão, que é `C:\work\temp\api-refs\V2\sdkfornet-api-ref_v2` para essas instruções.
4. Crie um atalho para o Google Chrome na `C:\work\temp\api-refs\V2` pasta. Tenha cuidado para não mover o aplicativo original.
5. Nas propriedades do novo atalho, defina os seguintes campos:

   - Alvo: `"<path to the Google Chrome application>" --disable-web-security --user-data-dir=C:\work\temp\api-refs\V2\data "C:\work\temp\api-refs\V2\sdkfornet-api-ref_v2\Index.html"`

Para o `--user-data-dir` argumento, use um nome de pasta que funcione para o seu ambiente. A pasta não precisa existir.

- Comece em: `C:\work\temp\api-refs\V2`

6. Dê ao atalho um nome apropriado.
7. Abra o atalho para ver a referência antiga da API.

Histórico do documento

A tabela a seguir descreve as alterações importantes feitas desde a versão mais recente do Guia do desenvolvedor do AWS SDK para .NET . Para receber notificações sobre atualizações dessa documentação, inscreva-se em um feed RSS.

Alteração	Descrição	Data
O que há de novo	Foi adicionada uma observação sobre a versão 2.0 da Ferramenta de AWS Implantação para CLI do.NET.	17 de outubro de 2025
AWS OpsWorks	AWS OpsWorks chegou ao fim da vida útil. Foi removido da documentação.	16 de setembro de 2025
O que há de novo	End-of-support foi anunciado para a V3 do AWS SDK para .NET.	21 de agosto de 2025
O que há de novo	Disponibilidade geral para o provedor de cache distribuído	02 de julho de 2025
Estrutura de processamento de mensagens	O AWS Message Processing Framework para.NET atingiu a disponibilidade geral!	5 de maio de 2025
O que há de novo	A versão 4 do AWS SDK para .NET está disponível ao público em geral! Para obter informações sobre como migrar seus aplicativos para a V4, consulte Migração para a versão 4.	28 de abril de 2025
Migrando para a versão 4 do AWS SDK para .NET	Informações de pré-visualização incluídas sobre a	3 de março de 2025

	migração de aplicativos criados com o AWS SDK para .NET para a versão 4.	
Integração AWS com o.NET Aspire no AWS SDK para .NET	Incluiu informações sobre como AWS foi integrado com o.NET Aspire no. AWS SDK para .NET	15 de fevereiro de 2025
Observabilidade	Anúncio da versão de disponibilidade geral da observabilidade	10 de fevereiro de 2025
O que há de novo	Foram adicionadas informaçõ es sobre o novo comportam ento padrão para proteção de integridade.	15 de janeiro de 2025
Migração do cliente de criptografia S3 V4	Foram adicionadas informaçõ es sobre a migração da V2 para a V4.	9 de dezembro de 2024
O que há de novo	Foram adicionadas informaçõ es sobre a quarta versão prévia da AWS SDK para .NET versão 4.	15 de novembro de 2024