Entwicklerhandbuch

AWS SDK für .NET (V4)

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS SDK für .NET (V4): Entwicklerhandbuch

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Die Marken und Handelsmarken von Amazon dürfen nicht in einer Weise in Verbindung mit nicht von Amazon stammenden Produkten oder Services verwendet werden, die geeignet ist, Kunden irrezuführen oder Amazon in irgendeiner Weise herabzusetzen oder zu diskreditieren. Alle anderen Marken, die nicht Eigentum von Amazon sind, sind Eigentum ihrer jeweiligen Inhaber, die mit Amazon verbunden oder nicht verbunden oder von Amazon gesponsert oder nicht gesponsert sein können.

Table of Contents

..... ix
Was ist der AWS SDK für .NET 1
Über diese Version 1
Wartung und Support für SDK-Hauptversionen 2
Häufige Anwendungsfälle 2
Weitere Themen in diesem Abschnitt 2
Verwandte AWS Tools 3
Tools für Windows PowerShell und Tools für PowerShell Core 3
Toolkit for VS Code 3
Toolkit for Visual Studio 3
Toolkit für Azure DevOps 4
Unterstützte Plattformen 4
.NET Core 4
.NET Standard 2.0 5
.NET Framework 4.5 5
.NET Framework 3.5 5
Portable Klassenbibliothek und Xamarin 5
Unity-Unterstützung 5
SDKs und Tool-Referenz 5
Geschichte der Revision 6
Was ist neu 6
Weitere Ressourcen 8
Erste Schritte 11
Installation und Konfiguration Ihrer Toolchain 11
Plattformübergreifende Entwicklung 11
Windows mit Visual Studio und.NET Core 12
Nächster Schritt 12
Authentifizierung mit AWS 12
Aktivieren und Konfigurieren von IAM Identity Center 13
Konfigurieren Sie das SDK für die Verwendung von IAM Identity Center. 13
Starten Sie eine AWS Access-Portal-Sitzung 14
Zusätzliche Informationen 15
Erstellen einer einfachen Anwendung 16
Einfache plattformübergreifende App 16

Einfache Windows-basierte App ... 22
Nächste Schritte ... 28
Konfigurieren ... 29
Ein neues Projekt starten ... 29
AWSSDK Pakete installieren mit NuGet ... 31
Verwendung NuGet über die Befehlszeile oder das Terminal ... 32
Verwenden NuGet im Visual Studio Solution Explorer ... 32
NuGet Von der Package Manager Console aus verwenden ... 33
Installieren Sie AWSSDK Baugruppen ohne NuGet ... 34
AWS Region ... 35
Erstellen Sie einen Service-Client mit einer bestimmten Region ... 36
Geben Sie eine Region für alle Service-Clients an ... 37
Auflösung der Region ... 38
Besondere Informationen über die Region China (Peking) ... 38
Besondere Informationen zu neuen Diensten AWS ... 39
Auflösung von Anmeldeinformationen und Profilen ... 39
Auflösung des Profils ... 40
Verwenden von Anmeldeinformationen für Verbundbenutzerkonten ... 41
Angeben von Rollen oder temporären Anmeldeinformationen ... 41
Verwendung von Proxy-Anmeldeinformationen ... 42
Wiederholungen und Timeouts ... 42
Wiederholversuche ... 43
Timeouts ... 45
Beobachtbarkeit ... 47
Weitere Ressourcen ... 48
Konfigurieren Sie ein `TelemetryProvider` ... 48
Metriken ... 50
Telemetrieanbieter ... 52
Benutzer und Rollen ... 53
Benutzer und Berechtigungssätze ... 54
Servicerollen ... 54
Erweiterte Konfiguration ... 55
AWSSDK.Erweiterungen. NETCore.Setup und IConfiguration ... 56
Konfigurieren von anderen Anwendungsparametern ... 60
Referenz zu Konfigurationsdateien für AWS SDK für .NET ... 69
Verwenden von Legacy-Anmeldeinformationen ... 81

Wichtige Warnhinweise und Richtlinien für Anmeldeinformationen 82
Die Datei mit den gemeinsam genutzten AWS Anmeldeinformationen verwenden 83
Den SDK Store verwenden (nur Windows) 87
Verwenden der SDK 92
Asynchrone Programmierung 92
Paginierung 94
Wo finde ich Paginatoren? 94
Was geben mir Paginatoren? 94
Synchrone und asynchrone Paginierung 95
Beispiel 95
Zusätzliche Überlegungen zu Paginatoren 99
Support für HTTP 2 100
Weitere Überlegungen 102
Zusätzliche Tools 102
AWS Tool bereitstellen 103
AWS Framework für die Nachrichtenverarbeitung für.NET 103
Integrationen mit.NET Aspire 103
Erweiterte Authentifizierung 104
Single Sign-On 104
Voraussetzungen 105
Ein SSO-Profil einrichten 105
SSO-Token generieren und verwenden 107
Weitere Ressourcen 112
Tutorials 112
Tutorial: Ausschließlich.NET-Anwendung 113
Tutorial: AWS CLI und .NET-Anwendung 122
Bereitstellen auf AWS 132
Über die .NET-CLI bereitstellen 132
Mit den IDE-Toolkits bereitstellen 132
Anwendungsfälle 133
ASP.NET Core-Apps 133
.NET-Konsolen-Apps 134
Blazor-Apps WebAssembly 135
AWS Lambda projekte 136
Voraussetzungen 136
Verfügbare Lambda-Befehle 137

Schritte zur Bereitstellung 137
AWS Dienste anrufen 139
Geführte Codebeispiele 139
AWS CloudFormation 140
Amazon Cognito 144
DynamoDB 153
Amazon EC2 183
IAM 246
Amazon S3 266
Amazon SNS 276
Amazon SQS 281
AWS Lambda 315
APIs 315
Voraussetzungen 315
Zusätzliche Informationen 315
Themen 315
Lambda-Anmerkungen 315
Bibliotheken und Frameworks auf hohem Niveau 317
Framework für die Nachrichtenverarbeitung 318
Integration AWS mit.NET Aspire 339
Andere Dienste und Konfiguration 340
Codebeispiele 341
Aurora 342
Grundlagen 344
Aktionen 368
Auto Scaling 384
Grundlagen 344
Aktionen 368
Amazon Bedrock 420
Aktionen 368
Amazon Bedrock Runtime 424
Anthropic Claude 424
Cohere Command 429
Meta Lama 433
Mistral KI 437
AWS CloudFormation 440

CloudWatch ... 444
Grundlagen ... 344
Aktionen ... 368
Amazon Cognito Identity Provider ... 499
Aktionen ... 368
AWS Control Tower ... 501
Grundlagen ... 344
Aktionen ... 368
DynamoDB ... 540
Grundlagen ... 344
Aktionen ... 368
Amazon EC2 ... 580
Amazon ECS ... 582
Amazon S3 ... 584
Aktionen ... 368
Szenarien ... 586
Migrieren Sie Ihr Projekt ... 594
Migration auf Version 4 ... 594
.NET Framework. ... 594
Wertetypen ... 595
Sammlungen ... 595
AWS Security Token Service (STS) ... 596
Änderungen im Zusammenhang mit `ClientConfig` ... 596
Die AWSSDK .Extensions. NETCore.Setup-Paket NuGet ... 597
`CookieSigner` und `UrlSigner` ... 598
DateTime im Vergleich zu UTC DateTime ... 598
DateTime Parsen ... 599
`ConvertFromUnixEpochSeconds` und `ConvertFromUnixEpochMilliseconds` ... 599
Protokollierung ... 599
Support für HTTP 2 ... 600
Single Sign-On ... 600
DynamoDB-spezifische Änderungen ... 601
Spezifische Änderungen für EC2 ... 605
Spezifische Änderungen für S3 ... 605
Programmierelemente, die entfernt wurden ... 609
Migration von .NET-Standard 1.3 ... 612

Sicherheit ... 614
Datenschutz ... 615
Identitäts- und Zugriffsverwaltung ... 616
Zielgruppe ... 616
Authentifizierung mit Identitäten ... 617
Verwalten des Zugriffs mit Richtlinien ... 618
Wie AWS-Services arbeiten Sie mit IAM ... 621
Fehlerbehebung bei AWS Identität und Zugriff ... 621
Compliance-Validierung ... 623
Ausfallsicherheit ... 623
Infrastruktursicherheit ... 624
Erzwingen einer Mindest-TLS-Version ... 625
.NET Core ... 625
.NET Framework. ... 626
AWS -Tools für PowerShell ... 627
Xamarin ... 628
Unity ... 629
Browser (für Blazor) WebAssembly ... 629
Migration des S3-Verschlüsselungsclients ... 629
Überblick über die Migration ... 630
Aktualisieren Sie bestehende Clients auf V1-Transitional-Clients, um neue Formate lesen zu können ... 630
Migrieren Sie V1-Transitiony-Clients zu V2-Clients, um neue Formate zu schreiben ... 631
Aktualisieren Sie V2-Clients so, dass sie V1-Formate nicht mehr lesen ... 635
Besondere Überlegungen ... 636
Beschaffung von AWSSDK Baugruppen ... 636
Laden Sie ZIP-Dateien herunter und entpacken Sie sie ... 636
Zugreifen auf Anmeldeinformationen und Profile in einer Anwendung ... 637
Beispiele für den Unterricht CredentialProfileStoreChain ... 638
Beispiele für Klassen SharedCredentialsFile und AWSCredentials Factory ... 639
Unity-Unterstützung ... 640
Xamarin-Unterstützung ... 641
API-Referenz ... 642
Über API-Referenzversionen ... 642
Dokumentverlauf ... 645

Version 4 (V4) von AWS SDK für .NET wurde veröffentlicht!

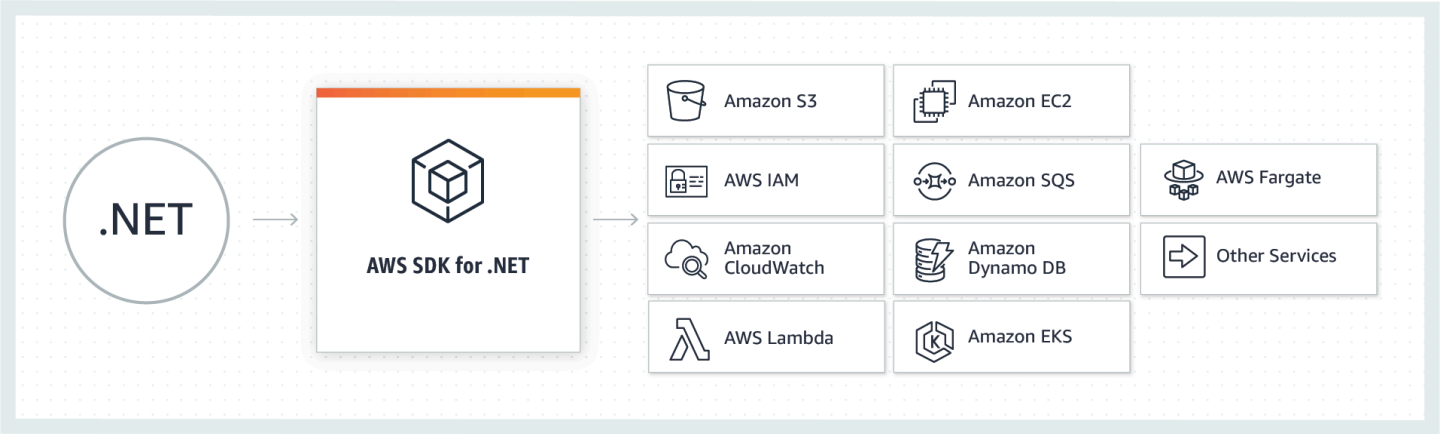
Informationen zu wichtigen Änderungen und zur Migration Ihrer Anwendungen finden Sie im Migrationsthema.

Click here for details

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

Was ist der AWS SDK für .NET

Das AWS SDK für .NET macht es einfacher, .NET-Anwendungen zu erstellen, die kostengünstige, skalierbare und zuverlässige AWS Dienste wie Amazon Simple Storage Service (Amazon S3) und Amazon Elastic Compute Cloud (Amazon EC2) nutzen. Das SDK vereinfacht die Nutzung von AWS Diensten, indem es eine Reihe von Bibliotheken bereitstellt, die für .NET-Entwickler konsistent und vertraut sind.



(OK, verstanden! Ich bin bereit, mich einzurichten und eine kurze Tour zu machen.)

Über diese Version

Aktuelle und frühere Dokumentation

Diese Dokumentation bezieht sich auf Version 4.0 und höher (V4) von AWS SDK für .NET. Sie konzentriert sich hauptsächlich auf .NET Core und ASP.NET Core, enthält aber auch Informationen zu.NET Framework und ASP.NET 4. x. Neben Windows und Visual Studio wird die plattformübergreifende Entwicklung gleichermaßen berücksichtigt.
Informationen zur Migration von früheren Versionen des SDK finden Sie unter. Migrieren Sie Ihr Projekt
Informationen zur vorherigen Version des SDK, V3, die in future veraltet sein wird, finden Sie im AWS SDK für .NET (V3) Developer Guide.
Informationen zu veralteten Inhalten für frühere Versionen des SDK finden Sie in den folgenden Artikeln:

- AWS SDK für .NET (Version 2, veraltet) Entwicklerhandbuch

- Veraltete API-Referenzen für AWS SDK für .NET

Wartung und Support für SDK-Hauptversionen

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

- AWS SDKs Richtlinien zur Wartung von Tools
- AWS SDKs und Matrix zur Unterstützung der Tools-Versionen

Häufige Anwendungsfälle

Das AWS SDK für .NET hilft Ihnen dabei, mehrere überzeugende Anwendungsfälle zu realisieren, darunter die folgenden:

- Verwalten Sie Benutzer und Rollen mit AWS Identity and Access Management (IAM).
- Greifen Sie auf Amazon Simple Storage Service (Amazon S3) zu, um Buckets zu erstellen und Objekte zu speichern.
- HTTP-Abonnements von Amazon Simple Notification Service (Amazon SNS) für Themen verwalten.
- Verwenden Sie das S3-Übertragungsdienstprogramm, um Dateien von Ihren Xamarin-Anwendungen nach Amazon S3 zu übertragen.
- Verwenden Sie Amazon Simple Queue Service (Amazon SQS), um Nachrichten und Workflows zwischen Komponenten in einem System zu verarbeiten.
- Führen Sie effiziente Amazon S3-Übertragungen durch, indem Sie SQL-Anweisungen an Amazon S3 Select senden.
- Erstellen und starten Sie EC2Amazon-Instances und konfigurieren und fordern Sie EC2 Amazon-Spot-Instances an.

Weitere Themen in diesem Abschnitt

- AWS Tools im Zusammenhang mit dem AWS SDK für .NET

- Plattformen, die unterstützt werden von AWS SDK für .NET
- AWS SDKs Referenzhandbuch für Tools und Tools
- Geschichte der Revision
- Was ist neu in der AWS SDK für .NET
- Weitere Ressourcen

AWS Tools im Zusammenhang mit dem AWS SDK für .NET

Tools für Windows PowerShell und Tools für PowerShell Core

Die PowerShell Module AWS Tools for Windows PowerShell und AWS Tools for PowerShell Core basieren auf den Funktionen von AWS SDK für .NET. Mit den AWS PowerShell Tools können Sie von der PowerShell Eingabeaufforderung aus Skripts für Operationen an Ihren AWS Ressourcen erstellen. Obwohl die Cmdlets mithilfe der Dienstclients und Methoden aus dem SDK implementiert werden, bieten die Cmdlets eine idiomatische Oberfläche für die Angabe von Parametern und die PowerShell Verarbeitung von Ergebnissen.

Um zu beginnen, sehen Sie sich AWS Tools for Windows PowerShell an.

Toolkit for VS Code

Das AWS Toolkit for Visual Studio Code ist ein Plugin für den Visual Studio-Code (VS-Code)-Editor. Das Toolkit erleichtert das Entwickeln, Debuggen und Bereitstellen von Anwendungen, die AWS verwenden.

Mit dem Toolkit können Sie Folgendes tun:

- Erstellen Sie serverlose Anwendungen, die AWS Lambda Funktionen enthalten, und stellen Sie die Anwendungen dann auf einem AWS CloudFormation Stack bereit.
- Arbeiten Sie mit EventBridge Amazon-Schemas.
- IntelliSense Wird bei der Arbeit mit Amazon ECS-Aufgabendefinitionsdateien verwendet.
- Visualisieren Sie eine Anwendung AWS Cloud Development Kit (AWS CDK) .

Toolkit for Visual Studio

Das AWS Toolkit for Visual Studio ist ein Plug-in für die Visual Studio-IDE, das Ihnen das Entwickeln, Debuggen und Bereitstellen von.NET-Anwendungen, die Amazon Web Services verwenden,

erleichtert. Das Toolkit for Visual Studio bietet Visual Studio-Vorlagen für Dienste wie Lambda und Bereitstellungsassistenten für Webanwendungen und serverlose Anwendungen. Sie können den AWS Explorer verwenden, um EC2 Amazon-Instances zu verwalten, mit Amazon DynamoDB-Tabellen zu arbeiten, Nachrichten in Amazon Simple Notification Service (Amazon SNS) - Warteschlangen zu veröffentlichen und vieles mehr — alles in Visual Studio.

Informationen zu den ersten Schritten finden Sie unter Einrichten von. AWS Toolkit for Visual Studio

Toolkit für Azure DevOps

Das AWS Toolkit for Microsoft Azure DevOps fügt Aufgaben hinzu, um auf einfache Weise die Erstellung und Veröffentlichung von Pipelines in Azure DevOps und Azure DevOps Server für die Zusammenarbeit mit AWS Diensten zu ermöglichen. Sie können mit Amazon S3,, AWS Elastic Beanstalk, Lambda AWS CodeDeploy AWS CloudFormation, Amazon Simple Queue Service (Amazon SQS) und Amazon SNS arbeiten. Sie können Befehle auch mit dem PowerShell Windows-Modul und dem AWS Command Line Interface () ausführen.AWS CLI

Informationen zu den ersten AWS Toolkit for Azure DevOps Schritten finden Sie im AWS Toolkit for Microsoft Azure DevOps Benutzerhandbuch.

Plattformen, die unterstützt werden von AWS SDK für .NET

Das AWS SDK für .NET bietet verschiedene Gruppen von Assemblys für Entwickler, die auf verschiedene Plattformen abzielen. Jedoch sind nicht alle SDK-Funktionalitäten auf jeder dieser Plattformen identisch. Dieses Thema beschreibt die Unterschiede in der Unterstützung für jede Plattform.

.NET Core

Das AWS SDK für .NET unterstützt Anwendungen, die für .NET Core geschrieben wurden (.NET Core 3.1, .NET 5, .NET 6 usw.). AWS Dienstclients unterstützen nur asynchrone Aufrufmuster in.NET Core. Dies wirkt sich auch auf viele der High-Level-Abstraktionen aus, die auf Service-Clients basieren, wie Amazon S3`TransferUtility`, das nur asynchrone Aufrufe in der .NET Core-Umgebung unterstützt.

.NET Standard 2.0

Nicht-Framework-Varianten von AWS SDK für .NET entsprechen dem.NET Standard 2.0. Das AWS SDK für .NET stellt nur asynchrone Methoden für Anwendungen bereit, die für .NET Standard geschrieben wurden.

.NET Framework 4.5

Diese Version von wurde für .NET Framework 4.7.2 kompiliert und läuft in der.NET 4.0-Laufzeit. AWS SDK für .NET AWS Service-Clients unterstützen synchrone und asynchrone Aufrufmuster und verwenden die in C# 5.0 eingeführten Schlüsselwörter async und await.

.NET Framework 3.5

Version 4 von unterstützt AWS SDK für .NET nicht.NET Framework 3.5.

Portable Klassenbibliothek und Xamarin

Das enthält AWS SDK für .NET auch eine Implementierung der Portable Class Library. Die Implementierung der Portable Class Library kann auf mehrere Plattformen abzielen, darunter Universal Windows Platform (UWP) und Xamarin auf iOS und Android. Weitere Informationen finden Sie im Mobile SDK for .NET and Xamarin. AWS Service-Clients unterstützen nur asynchrone Aufrufmuster.

Unity-Unterstützung

Informationen zur Unity-Unterstützung finden Sie unterBesondere Überlegungen zur Unity-Unterstützung.

AWS SDKs Referenzhandbuch für Tools und Tools

Das Referenzhandbuch AWS SDKs und die Tools enthalten Informationen, die für viele der Toolkits AWS SDKs und die relevant und wichtig sind. AWS CLI Im Folgenden finden Sie einige Beispiele für die Informationen, die die Referenz enthält:

- Informationen über die geteilten `credentials` Dateien AWS`config` und deren Speicherort.
- AWS Konten, Benutzer und Rollen einrichten
- Referenz zu den Einstellungen für Konfiguration und Authentifizierung

- AWS Common Runtime (CRT) -Bibliotheken
- AWS SDKs Richtlinien zur Wartung von Tools
- AWS SDKs und Matrix zur Unterstützung der Tools-Versionen

Geschichte der Revision

Im Folgenden erfahren Sie, was sich in den verschiedenen Versionen geändert hat:

- SDK-Änderungsprotokolle
- Was ist neu in der AWS SDK für .NET
- Dokumentverlauf

Was ist neu in der AWS SDK für .NET

Allgemeine Informationen zu neuen Entwicklungen im Zusammenhang mit finden Sie auf der Produktseite unter https://aws.amazon.com/sdk-for-net/und in den SDK-Änderungsprotokollen. AWS SDK für .NET

Folgendes ist neu in der AWS SDK für .NET.

17. Oktober 2025: Version 2.0 des AWS Deploy Tools

Version 2.0 des AWS Deploy Tools for .NET CLI wurde veröffentlicht. Informationen zum Deploy Tool finden Sie unterStellen Sie Anwendungen bereit für AWS. Informationen zur Version 2.0 des Deploy Tools finden Sie im Blogbeitrag Was ist neu im AWS Deploy Tool for .NET.

21. August 2025: Demnächst end-of-support für Version 3 des AWS SDK für .NET

Die end-of-support für Version 3 (V3) von AWS SDK für .NET wurde angekündigt. Informationen zur Migration Ihrer V3-Anwendungen und zur Vermeidung von Unterbrechungen finden Sie im Migrationsleitfaden. Weitere Informationen finden Sie im Blogbeitrag Ankündigung der end-of-support Version 3. AWS SDK für .NET

2. Juli 2025: Allgemeine Verfügbarkeit des Distributed Cache Providers

AWS Der.NET Distributed Cache Provider für Amazon DynamoDB ist allgemein verfügbar! Dieser Anbieter implementiert die ASP.NET Core IDistributedCache-Schnittstelle, sodass Sie die vollständig

verwaltete und dauerhafte Infrastruktur von DynamoDB mit minimalen Codeänderungen in Ihre Caching-Ebene integrieren können. Der Distributed Cache Provider ist über AWS verfügbar. AspNetCore. DistributedCacheProviderPaket auf NuGet. Weitere Informationen finden Sie im Blogbeitrag. AWS NET Distributed Cache Provider for Amazon DynamoDB now General Available.

> **Note**
>
> Die veröffentlichte Version des Distributed Cache Providers wird nur auf Version 4 von unterstützt. AWS SDK für .NET Wenn Sie die Vorschauversion des Distributed Cache Provider ausprobiert haben, müssen Sie möglicherweise die Paketabhängigkeiten in Ihrer Anwendung aktualisieren.

5. Mai 2025: Allgemeine Verfügbarkeit des AWS Message Processing Framework für.NET

Das AWS Message Processing Framework für.NET ist allgemein verfügbar! Das Framework ist ein AWS natives Framework, das die Entwicklung von.NET-Nachrichtenverarbeitungsanwendungen vereinfacht, die AWS Dienste wie Amazon Simple Queue Service (SQS), Amazon Simple Notification Service (SNS) und Amazon verwenden. EventBridge

28. April 2025: Version 4 des AWS SDK für .NET

Version 4 von AWS SDK für .NET ist allgemein verfügbar! Informationen zur Migration Ihrer Anwendungen auf V4 finden Sie unterMigration auf Version 4. Lesen Sie auch den Blogbeitrag Allgemeine Verfügbarkeit von AWS SDK für .NET V4.0, in dem die allgemeine Verfügbarkeit angekündigt wird.

15. Februar 2025: Integrationen mit.NET Aspire

Integrationen mit.NET Aspire zur Verbesserung der inneren Entwicklungsschleife wurden veröffentlicht. Weitere Informationen finden Sie unter Integration AWS mit .NET Aspire in AWS SDK für .NET.

10. Februar 2025: GA-Version für Beobachtbarkeit

Beobachtbarkeit ist das Ausmaß, in dem aus den von ihm ausgegebenen Daten auf den aktuellen Zustand eines Systems geschlossen werden kann. Beobachtbarkeit wurde hinzugefügt AWS SDK für .NET, einschließlich der Implementierung eines Telemetrieanbieters. Weitere Informationen finden Sie Beobachtbarkeit in diesem Handbuch und im Blogbeitrag Ankündigung der allgemeinen Verfügbarkeit von.NET-Bibliotheken. AWS OpenTelemetry

15. Januar 2025: Neues Standardverhalten für den Integritätsschutz

Ab Version 3.7.412.0 von bietet das SDK standardmäßige Integritätsschutzmaßnahmen AWS SDK für .NET, indem es automatisch eine Prüfsumme für Uploads berechnet. `CRC32` Weitere Informationen finden Sie in der Ankündigung unter. GitHub https://github.com/aws/aws-sdk-net/issues/3610 Das SDK bietet auch globale Einstellungen für den Schutz der Datenintegrität, die Sie extern festlegen können. Weitere Informationen finden Sie unter Datenintegritätsschutz im Referenzhandbuch AWS SDKs und im Tools-Referenzhandbuch.

15. November 2024: Blogbeitrag zur Vorschauversion 4 für Version 4

Vorschau 4 der AWS SDK für .NET Version 4 wurde am 15. November 2024 veröffentlicht. Weitere Informationen zu dieser Vorschau finden Sie im Blogbeitrag Preview 4 of AWS SDK für .NET V4.

16. August 2024: Blogbeitrag zur Vorschauversion 1 für Version 4

Die AWS SDK für .NET Version 4 wurde am 16. August 2024 als erste Vorschau veröffentlicht. Weitere Informationen zu dieser Vorschau finden Sie im Blogbeitrag Preview 1 of AWS SDK für .NET V4.

Weitere Ressourcen

Unterstützte Dienste

Das AWS SDK für .NET unterstützt die meisten AWS Infrastrukturprodukte, und es werden regelmäßig weitere Dienste hinzugefügt. Eine Liste der vom SDK unterstützten AWS Dienste finden Sie in der SDK-README-Datei.

Homepage für AWS SDK für .NET

Weitere Informationen zum AWS SDK für .NET finden Sie auf der Homepage des SDK unter https://aws.amazon.com/sdk-for-net/.

SDK-Referenzdokumentation

Die SDK-Referenzdokumentation bietet Ihnen die Möglichkeit, den gesamten im SDK enthaltenen Code zu durchsuchen und zu durchsuchen. Sie enthält eine ausführliche Dokumentation und Anwendungsbeispiele. Weitere Informationen finden Sie in der AWS SDK für .NET -API-Referenz.

AWS Re:post (früher AWS Foren)

Besuchen Sie AWS re:POST, speziell das Thema für AWS SDK für .NET, um Fragen zu stellen oder Feedback zu geben. AWS Auf jeder Dokumentationsseite befindet sich unten auf der Seite ein Link zum Testen von AWS re:POST, über den Sie zum entsprechenden re:POST-Thema gelangen. AWS Techniker beobachten die Themen und beantworten Fragen, Feedback und Probleme.

Wenn du bei re:POST angemeldet bist, kannst du auch einem Thema folgen. Um dem Thema für zu folgen AWS SDK für .NET, gehe auf die Seite „Alle Themen“, suche nach „.NET on AWS“ und wähle die Schaltfläche „Folgen“.

Toolkits

- AWS Toolkit for Visual Studio: Wenn Sie die Microsoft Visual Studio IDE verwenden, sollten Sie sich das AWS Toolkit for Visual Studio Benutzerhandbuch ansehen.
- AWS Toolkit for Visual Studio Code: Wenn Sie die Microsoft Visual Studio IDE verwenden, sollten Sie sich das AWS Toolkit for Visual Studio Code Benutzerhandbuch ansehen.

Hilfreiche Bibliotheken, Erweiterungen und Tools

In den aws-sdk-net Repositorien aws/dotnet und aws/ auf der GitHub Website finden Sie Links zu Bibliotheken, Tools und Ressourcen, die Sie zur Erstellung von.NET-Anwendungen und -Diensten verwenden können. AWS

Im Folgenden sind einige Beispiele aufgeführt:

- AWS .NET-Konfigurationserweiterung für Systems Manager
- AWS Erweiterungen.NET Core-Setup
- AWS .NET protokollieren
- Amazon Cognito Authentication Extension Library
- AWS X-Ray SDK for .NET

Sonstige Ressourcen

Im Folgenden finden Sie weitere Ressourcen, die sich als nützlich erweisen könnten:

- Entwicklernetzwerk
- .NET-Entwicklungsumgebung in der AWS Cloud — Schnellstart-Referenzbereitstellung
- Hallo, Cloud! Blog

- AWS Whitepaper: Entwicklung und Bereitstellung von.NET-Anwendungen auf AWS
- AWS Microservice Extractor for .NET
- Portierungsassistent für .NET
- AWS SDKs und Referenzhandbuch für Tools

Erste Schritte mit dem AWS SDK für .NET

Erfahren Sie, wie Sie das installieren, einrichten und verwenden AWS SDK für .NET. Sie müssen Ihre Toolchain installieren und eine Reihe wichtiger Dinge einrichten, die Ihre Anwendung für den Zugriff auf AWS Dienste benötigt. Dazu zählen:

- Ein geeignetes Benutzerkonto oder eine entsprechende Rolle
- Authentifizierungsinformationen für dieses Benutzerkonto oder für die Übernahme dieser Rolle

Informationen zur Konfiguration von Projekten und anderen wichtigen Komponenten finden Sie unterKonfiguration der AWS SDK für .NET.

Themen

- Installation und Konfiguration Ihrer Toolchain für die AWS SDK für .NET
- Authentifizieren des mit AWS SDK für .NETAWS
- Erstellen einer einfachen Anwendung mit dem AWS SDK für .NET

Installation und Konfiguration Ihrer Toolchain für die AWS SDK für .NET

Um die verwenden zu können AWS SDK für .NET, müssen bestimmte Entwicklungstools installiert sein.

Plattformübergreifende Entwicklung

Folgendes ist für die plattformübergreifende .NET-Entwicklung unter Windows, Linux oder macOS erforderlich:

- Microsoft .NET Core SDK, Version 2.1, 3.1 oder höher, einschließlich der .NET-Befehlszeilenschnittstelle (CLI) (**`dotnet`**) und der .NET Core-Runtime.

- Ein Code-Editor oder eine integrierte Entwicklungsumgebung (IDE), die für Ihr Betriebssystem und Ihre Anforderungen geeignet ist. Dies ist in der Regel eine, die eine gewisse Unterstützung für.NET Core bietet.

Beispiele hierfür sind Microsoft Visual Studio Code (VS Code), JetBrains Rider und Microsoft Visual Studio.

- (Optional) Ein AWS Toolkit, sofern eines für den ausgewählten Editor und Ihr Betriebssystem verfügbar ist.

  Zu den Beispielen gehören AWS Toolkit for Visual Studio CodeAWS Toolkit for JetBrains, und AWS Toolkit for Visual Studio.

Windows mit Visual Studio und.NET Core

Folgendes ist für die Entwicklung unter Windows mit Visual Studio und.NET Core erforderlich:

- Microsoft Visual Studio

- Microsoft .NET Core 2.1, 3.1 oder höher

  Dies ist normalerweise standardmäßig enthalten, wenn eine aktuelle Version von Visual Studio installiert wird.

- (Optional) Das AWS Toolkit for Visual Studio, ein Plug-in, das eine Benutzeroberfläche für die Verwaltung Ihrer AWS Ressourcen und lokalen Profile von Visual Studio aus bereitstellt. Informationen zur Installation des Toolkits finden Sie unter Einrichten von. AWS Toolkit for Visual Studio

  Weitere Informationen finden Sie im AWS Toolkit for Visual Studio -Benutzerhandbuch.

Nächster Schritt

Authentifizieren des mit AWS SDK für .NETAWS

Authentifizieren des mit AWS SDK für .NETAWS

Sie müssen festlegen, wie sich Ihr Code AWS bei der Entwicklung mit authentifiziert. AWS-Services Je nach Umgebung und verfügbarem Zugriff gibt es verschiedene Möglichkeiten, den programmatischen AWS Zugriff auf AWS Ressourcen zu konfigurieren.

Informationen zu den verschiedenen Authentifizierungsmethoden für das SDK finden Sie unter Authentifizierung und Zugriff im AWS SDKs Referenzhandbuch zu Tools.

In diesem Thema wird davon ausgegangen, dass ein neuer Benutzer sich lokal weiterentwickelt, von seinem Arbeitgeber noch keine Authentifizierungsmethode erhalten hat und diese AWS IAM Identity Center zum Abrufen temporärer Anmeldeinformationen verwenden wird. Wenn Ihre Umgebung nicht unter diese Annahmen fällt, treffen einige der Informationen in diesem Thema möglicherweise nicht auf Sie zu, oder einige der Informationen wurden Ihnen möglicherweise bereits gegeben.

Die Konfiguration dieser Umgebung erfordert mehrere Schritte, die sich wie folgt zusammenfassen lassen:

1. Aktivieren und Konfigurieren von IAM Identity Center
2. Konfigurieren Sie das SDK für die Verwendung von IAM Identity Center.
3. Starten Sie eine AWS Access-Portal-Sitzung

Aktivieren und Konfigurieren von IAM Identity Center

Um IAM Identity Center verwenden zu können, muss es zuerst aktiviert und konfiguriert werden. Einzelheiten dazu, wie Sie dies für das SDK tun können, finden Sie in Schritt 1 des Themas zur IAM Identity Center-Authentifizierung im Referenzhandbuch AWS SDKs und im Tools-Referenzhandbuch. Folgen Sie insbesondere den Anweisungen unter Ich habe keinen Zugriff über IAM Identity Center eingerichtet.

Konfigurieren Sie das SDK für die Verwendung von IAM Identity Center.

Informationen zur Konfiguration des SDK für die Verwendung von IAM Identity Center finden Sie in Schritt 2 des Themas zur IAM Identity Center-Authentifizierung im ReferenzhandbuchAWS SDKs zu Tools. Nachdem Sie diese Konfiguration abgeschlossen haben, sollte Ihr System die folgenden Elemente enthalten:

- Das AWS CLI, mit dem Sie eine AWS Access-Portal-Sitzung starten, bevor Sie Ihre Anwendung ausführen.
- Die gemeinsam genutzte AWS `config` Datei, die ein `[default]`Profil mit einer Reihe von Konfigurationswerten enthält, auf die vom SDK aus verwiesen werden kann. Informationen zum Speicherort dieser Datei finden Sie unter Speicherort der gemeinsam genutzten Dateien im Referenzhandbuch AWS SDKs und im Tools-Referenzhandbuch. Der AWS SDK für .NET

verwendet den SSO-Token-Anbieter des Profils, um Anmeldeinformationen abzurufen, bevor Anfragen an gesendet AWS werden. Der Wert `sso_role_name`, bei dem es sich um eine IAM-Rolle handelt, die mit einem Berechtigungssatz von IAM Identity Center verbunden ist, sollte den Zugriff auf die in Ihrer Anwendung verwendeten AWS-Services ermöglichen.

Die folgende `config`-Beispieldatei zeigt ein Standardprofil, das mit dem SSO-Token-Anbieter eingerichtet wurde. Die `sso_session`-Einstellung des Profils bezieht sich auf den benannten `sso-session`-Abschnitt. Der `sso-session` Abschnitt enthält Einstellungen zum Initiieren einer AWS Access-Portal-Sitzung.

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

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

Important

Wenn Sie die Authentifizierung verwenden AWS IAM Identity Center , muss Ihre Anwendung auf die folgenden NuGet Pakete verweisen, damit die SSO-Auflösung funktioniert:

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

Wenn Sie auf diese Pakete nicht verweisen, wird eine Laufzeitausnahme ausgelöst.

Starten Sie eine AWS Access-Portal-Sitzung

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

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

```
aws sso login
```

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

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

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

In der Antwort auf diesen Befehl sollten das in der freigegebenen `config`-Datei konfigurierte IAM-Identity-Center-Konto und der Berechtigungssatz angegeben werden.

> **Note**
>
> Wenn Sie bereits über eine aktive AWS Access-Portal-Sitzung verfügen und diese ausführen`aws sso login`, müssen Sie keine Anmeldeinformationen angeben. Beim Anmeldevorgang werden Sie möglicherweise aufgefordert, den AWS CLI Zugriff auf Ihre Daten zu gewähren. Da AWS CLI das auf dem SDK für Python aufbaut, können Berechtigungsnachrichten Variationen des `botocore` Namens enthalten.

Zusätzliche Informationen

- Weitere Informationen zur Verwendung von IAM Identity Center und SSO in einer Entwicklungsumgebung finden Sie Single Sign-On im Erweiterte Authentifizierung Abschnitt. Diese Informationen umfassen alternative und fortgeschrittenere Methoden sowie Tutorials, die Ihnen zeigen, wie Sie diese Methoden verwenden.
- Weitere Authentifizierungsoptionen für das SDK, wie z. B. die Verwendung von Profilen und Umgebungsvariablen, finden Sie im Kapitel Konfiguration im AWS SDKs Referenzhandbuch zu Tools.
- Weitere Informationen zu bewährten Methoden finden Sie unter Bewährte Methoden für die Sicherheit in IAM im IAM-Benutzerhandbuch.

- Informationen zum Erstellen kurzfristiger AWS Anmeldeinformationen finden Sie unter Temporäre Sicherheitsanmeldedaten im IAM-Benutzerhandbuch.
- Weitere Informationen zu anderen Anbietern von Anmeldeinformationen finden Sie unter Standardisierte Anmeldeinformationsanbieter im Referenzhandbuch AWS SDKs und im Tools-Referenzhandbuch.

Erstellen einer einfachen Anwendung mit dem AWS SDK für .NET

Dieser Abschnitt enthält grundlegende Tutorials für Entwickler, die noch nicht mit dem vertraut sind AWS SDK für .NET.

> **Note**
>
> Bevor Sie diese Tutorials verwenden können, müssen Sie zuerst Ihre Toolchain installiert und die SDK-Authentifizierung konfiguriert haben.

Informationen zur Entwicklung von Software für bestimmte AWS Dienste sowie Codebeispiele finden Sie unter AWS Dienste anrufen. Weitere Codebeispiele finden Sie unterSDK für .NET (v4) Codebeispiele.

Themen

- Einfache plattformübergreifende Anwendung mit dem AWS SDK für .NET
- Einfache Windows-basierte Anwendung mit dem AWS SDK für .NET
- Nächste Schritte

Einfache plattformübergreifende Anwendung mit dem AWS SDK für .NET

In diesem Tutorial werden und.NET Core für die AWS SDK für .NET plattformübergreifende Entwicklung verwendet. Das Tutorial zeigt Ihnen, wie Sie das SDK verwenden, um die Amazon S3 S3-Buckets aufzulisten, die Sie besitzen, und optional einen Bucket zu erstellen.

Sie führen dieses Tutorial mithilfe plattformübergreifender Tools wie der .NET-Befehlszeilenschnittstelle (CLI) aus. Weitere Möglichkeiten zur Konfiguration Ihrer Entwicklungsumgebung finden Sie unterInstallation und Konfiguration Ihrer Toolchain für die AWS SDK für .NET.

Erforderlich für plattformübergreifende .NET-Entwicklung unter Windows, Linux oder macOS:

- Microsoft .NET Core SDK, Version 2.1, 3.1 oder höher, einschließlich der .NET-Befehlszeilenschnittstelle (CLI) (**`dotnet`**) und der .NET Core-Runtime.

- Ein Code-Editor oder eine integrierte Entwicklungsumgebung (IDE), die für Ihr Betriebssystem und Ihre Anforderungen geeignet ist. Dies ist in der Regel eine, die eine gewisse Unterstützung für.NET Core bietet.

  Beispiele hierfür sind Microsoft Visual Studio Code (VS Code), JetBrains Rider und Microsoft Visual Studio.

> **Note**
>
> Bevor Sie diese Tutorials verwenden können, müssen Sie zuerst Ihre Toolchain installiert und die SDK-Authentifizierung konfiguriert haben.

Schritte

- Erstellen des Projekts
- Erstellen des Codes
- Führen Sie die Anwendung aus.
- Bereinigen

Erstellen des Projekts

1. Öffnen Sie die Befehlszeile oder das Terminal. Suchen oder erstellen Sie einen Betriebssystemordner, unter dem Sie ein .NET-Projekt erstellen können.
2. Führen Sie in diesem Ordner den folgenden Befehl aus, um das .NET-Projekt zu erstellen.

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

3. Gehen Sie zu dem neu erstellten `S3CreateAndList` Ordner und führen Sie die folgenden Befehle aus:

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

Mit den vorherigen Befehlen werden die NuGet Pakete über den NuGet Paketmanager installiert. Da wir genau wissen, welche NuGet Pakete wir für dieses Tutorial benötigen, können wir diesen Schritt jetzt ausführen. Es ist auch üblich, dass die benötigten Pakete während der Entwicklung bekannt werden. Wenn dies geschieht, kann zu diesem Zeitpunkt ein ähnlicher Befehl ausgeführt werden.

Erstellen des Codes

1. Suchen Sie `Program.cs` im `S3CreateAndList`-Ordner und öffnen Sie es in Ihrem Code-Editor.
2. Ersetzen Sie den Inhalt durch den folgenden Code und speichern Sie die Datei.

```
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;
        }
    }
}
```

Führen Sie die Anwendung aus.

1. Führen Sie den folgenden Befehl aus.

```
dotnet run
```

2. Untersuchen Sie die Ausgabe, um die Anzahl der Amazon S3 S3-Buckets, die Sie besitzen, falls vorhanden, und deren Namen zu sehen.
3. Wählen Sie einen Namen für einen neuen Amazon S3 S3-Bucket. Verwenden Sie "dotnet-quicktour-s3-1-cross-“ als Basis und fügen Sie etwas Einzigartiges hinzu, z. B. eine GUID oder Ihren Namen. Beachten Sie unbedingt die Regeln für Bucket-Namen, wie sie unter Regeln für die Bucket-Benennung im Amazon S3 S3-Benutzerhandbuch beschrieben sind.
4. Führen Sie den folgenden Befehl aus und *amzn-s3-demo-bucket* ersetzen Sie ihn durch den Namen des von Ihnen ausgewählten Buckets.

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

5. Untersuchen Sie die Ausgabe, um den neuen Bucket anzuzeigen, der erstellt wurde.

Bereinigen

Während der Durchführung dieses Tutorials haben Sie einige Ressourcen erstellt, die Sie zu diesem Zeitpunkt bereinigen können.

- Wenn Sie den Bucket, den die Anwendung in einem früheren Schritt erstellt hat, nicht behalten möchten, löschen Sie ihn mithilfe der Amazon S3 S3-Konsole unter https://console.aws.amazon.com/s3/.
- Wenn Sie Ihr .NET-Projekt nicht beibehalten möchten, entfernen Sie den `S3CreateAndList`-Ordner aus Ihrer Entwicklungsumgebung.

Nächste Schritte

Kehren Sie zum Schnell-Tour-Menü zurück oder fahren Sie direkt zum Ende dieser Kurztour fort.

Einfache Windows-basierte Anwendung mit dem AWS SDK für .NET

In diesem Tutorial wird das AWS SDK für .NET unter Windows mit Visual Studio und.NET Core verwendet. Das Tutorial zeigt Ihnen, wie Sie das SDK verwenden, um die Amazon S3 S3-Buckets aufzulisten, die Sie besitzen, und optional einen Bucket zu erstellen.

Sie führen dieses Tutorial unter Windows mithilfe von Visual Studio und .NET Core aus. Weitere Möglichkeiten zur Konfiguration Ihrer Entwicklungsumgebung finden Sie unterInstallation und Konfiguration Ihrer Toolchain für die AWS SDK für .NET.

Für die Entwicklung unter Windows mit Visual Studio und.NET Core erforderlich:

- Microsoft Visual Studio

- Microsoft .NET Core 2.1, 3.1 oder höher

  Dies ist normalerweise standardmäßig enthalten, wenn eine aktuelle Version von Visual Studio installiert wird.

> **Note**
>
> Bevor Sie diese Tutorials verwenden können, müssen Sie zuerst Ihre Toolchain installiert und die SDK-Authentifizierung konfiguriert haben.

Schritte

- Erstellen des Projekts
- Erstellen des Codes
- Führen Sie die Anwendung aus.
- Bereinigen

Erstellen des Projekts

1. Öffnen Sie Visual Studio und erstellen Sie ein neues Projekt, das die C#-Version der Konsolen-App-Vorlage verwendet, d. h. mit der Beschreibung: „... zum Erstellen einer Befehlszeilenanwendung, die auf.NET ausgeführt werden kann...“. Benennen Sie das Projekt `S3CreateAndList`.

   > **Note**
   >
   > Wählen Sie nicht die. NET Framework-Version der Konsolen-App-Vorlage aus, und wenn ja, stellen Sie sicher, dass Sie.NET Framework 4.7.2 oder höher verwenden.

2. Wenn das neu erstellte Projekt geladen ist, wählen Sie Tools, NuGetPackage Manager, Manage NuGet Packages for Solution.
3. Suchen Sie nach den folgenden NuGet Paketen und installieren Sie sie im Projekt: `AWSSDK.S3AWSSDK.SecurityToken`,`AWSSDK.SSO`, und `AWSSDK.SSOOIDC`

   Bei diesem Vorgang werden die NuGet Pakete über den NuGet Paketmanager installiert. Da wir genau wissen, welche NuGet Pakete wir für dieses Tutorial benötigen, können wir diesen Schritt jetzt ausführen. Es ist auch üblich, dass die benötigten Pakete während der Entwicklung bekannt werden. Wenn dies geschieht, folgen Sie einem ähnlichen Prozess, um sie zum jeweiligen Zeitpunkt zu installieren.

4. Wenn Sie beabsichtigen, die Anwendung von der Befehlszeile aus auszuführen, öffnen Sie jetzt eine Befehlszeile und navigieren Sie zu dem Ordner, der die Build-Ausgabe enthalten soll. Das ist in der Regel so`S3CreateAndList\S3CreateAndList\bin\Debug\net6.0`, hängt aber von Ihrer Umgebung ab.

Erstellen des Codes

1. Suchen Sie im `S3CreateAndList`-Projekt `Program.cs` und öffnen Sie es in der IDE.
2. Ersetzen Sie den Inhalt durch den folgenden Code und speichern Sie die Datei.

```
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. Erstellen Sie die Anwendung

> **Note**
>
> Wenn Sie eine ältere Version von Visual Studio verwenden, wird möglicherweise ein Buildfehler ähnlich dem folgenden angezeigt:
> „Die Funktion 'Async Main' ist in C# 7.0 nicht verfügbar. Bitte verwenden Sie die Sprachversion 7.1 oder höher.“
> Wenn Sie diesen Fehler erhalten, richten Sie Ihr Projekt so ein, dass es eine neuere Version der Sprache verwendet. Dies erfolgt in der Regel in den Projekteigenschaften Build, Advanced.

Führen Sie die Anwendung aus.

1. Führen Sie die Anwendung ohne Befehlszeilenargumente aus. Tun Sie dies entweder in der Befehlszeile (falls Sie zuvor eine geöffnet haben) oder in der IDE.
2. Untersuchen Sie die Ausgabe, um die Anzahl der Amazon S3 S3-Buckets, die Sie besitzen, falls vorhanden, und deren Namen zu sehen.
3. Wählen Sie einen Namen für einen neuen Amazon S3 S3-Bucket. Verwenden Sie "dotnet-quicktour-s3-1-winvs-“ als Basis und fügen Sie etwas Einzigartiges hinzu, z. B. eine GUID oder Ihren Namen. Beachten Sie unbedingt die Regeln für Bucket-Namen, wie sie unter Regeln für die Bucket-Benennung im Amazon S3 S3-Benutzerhandbuch beschrieben sind.
4. Führen Sie die Anwendung erneut aus und geben Sie diesmal den Bucket-Namen an.

Ersetzen Sie *amzn-s3-demo-bucket* in der Befehlszeile den folgenden Befehl durch den Namen des Buckets, den Sie ausgewählt haben.

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

Oder, wenn Sie die Anwendung in der IDE ausführen, wählen Sie Project, S3 CreateAndList Properties, Debug und geben Sie dort den Bucket-Namen ein.

5. Untersuchen Sie die Ausgabe, um den neuen Bucket anzuzeigen, der erstellt wurde.

Bereinigen

Während der Durchführung dieses Tutorials haben Sie einige Ressourcen erstellt, die Sie zu diesem Zeitpunkt bereinigen können.

- Wenn Sie den Bucket, den die Anwendung in einem früheren Schritt erstellt hat, nicht behalten möchten, löschen Sie ihn mithilfe der Amazon S3 S3-Konsole unter https://console.aws.amazon.com/s3/.
- Wenn Sie Ihr .NET-Projekt nicht beibehalten möchten, entfernen Sie den `S3CreateAndList`-Ordner aus Ihrer Entwicklungsumgebung.

Nächste Schritte

Kehren Sie zum Schnell-Tour-Menü zurück oder fahren Sie direkt zum Ende dieser Kurztour fort.

Nächste Schritte

Stellen Sie sicher, dass Sie alle Ressourcen bereinigen, die Sie während der Durchführung dieser Tutorials erstellt haben. Dies können AWS Ressourcen oder Ressourcen in Ihrer Entwicklungsumgebung sein, z. B. Dateien und Ordner.

Nachdem Sie das besichtigt haben AWS SDK für .NET, möchten Sie vielleicht mit Ihrem Projekt beginnen.

Konfiguration der AWS SDK für .NET

Um das verwenden zu können AWS SDK für .NET, müssen Sie Ihre Umgebung einrichten und außerdem eine Reihe wichtiger Dinge konfigurieren, die Ihre Anwendung für den Zugriff auf AWS Dienste benötigt. Dazu zählen:

- Spezifikation der AWS Region
- AWSSDK Pakete oder Baugruppen

Einige der Themen in diesem Abschnitt enthalten Informationen zur Konfiguration dieser wichtigen Dinge.

Andere Themen in diesem Abschnitt und in anderen Abschnitten enthalten Informationen zu fortgeschritteneren Methoden zur Konfiguration Ihres Projekts.

Themen

- Ein neues Projekt starten
- AWSSDK Pakete installieren mit NuGet
- Installieren Sie AWSSDK Baugruppen ohne NuGet
- Festlegung der AWS Region für AWS SDK für .NET
- Auflösung von Anmeldeinformationen und Profilen
- Wiederholungen und Timeouts
- Beobachtbarkeit
- Zusätzliche Informationen über Benutzer und Rollen
- Erweiterte Konfiguration für Ihr AWS SDK für .NET Projekt
- Verwenden von Legacy-Anmeldeinformationen

Ein neues Projekt starten

Es gibt verschiedene Techniken, mit denen Sie ein neues Projekt starten können, um auf AWS Dienste zuzugreifen. Im Folgenden sind einige dieser Techniken aufgeführt:

- Wenn Sie mit der .NET-Entwicklung noch nicht vertraut sind AWS oder zumindest neu darin sind AWS SDK für .NET, finden Sie vollständige Beispiele unterErstellen einer einfachen Anwendung. Es gibt Ihnen eine Einführung in das SDK.

- Sie können ein Basisprojekt mithilfe der .NET CLI starten. Um ein Beispiel dafür zu sehen, öffnen Sie eine Befehlszeile oder ein Terminal, erstellen Sie einen Ordner oder ein Verzeichnis, navigieren Sie dorthin und geben Sie dann Folgendes ein.

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

  Es wird ein leeres Projekt erstellt, zu dem Sie Code und NuGet Pakete hinzufügen können. Weitere Informationen finden Sie im Leitfaden zu .NET Core.

  Verwenden Sie Folgendes, um eine Liste von Projektvorlagen zu sehen: `dotnet new --list`

- Das AWS Toolkit for Visual Studio beinhaltet C#-Projektvorlagen für eine Vielzahl von AWS Diensten. Nachdem Sie das Toolkit in Visual Studio installiert haben, können Sie beim Erstellen eines neuen Projekts auf die Vorlagen zugreifen.

  Informationen dazu finden Sie im AWS Toolkit for Visual Studio Benutzerhandbuch unter Arbeiten mit AWS Diensten. In mehreren der Beispiele in diesem Abschnitt werden neue Projekte erstellt.

- Wenn Sie mit Visual Studio unter Windows entwickeln, aber ohne AWS Toolkit for Visual Studio, verwenden Sie Ihre typischen Techniken zum Erstellen eines neuen Projekts.

  Um ein Beispiel zu sehen, öffnen Sie Visual Studio und wählen Sie Datei, Neu, Projekt. Suchen Sie nach „.net core“ und wählen Sie die C#-Version der Vorlage für die Konsolen-App (.NET Core) oder die WPF-App (.NET Core) aus. Es wird ein leeres Projekt erstellt, zu dem Sie Code und NuGet Pakete hinzufügen können.

Einige Beispiele für die Arbeit mit AWS Diensten finden Sie unterGeführte Codebeispiele.

> **Important**
>
> Wenn Sie die Authentifizierung verwenden AWS IAM Identity Center , muss Ihre Anwendung auf die folgenden NuGet Pakete verweisen, damit die SSO-Auflösung funktioniert:

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

Wenn Sie auf diese Pakete nicht verweisen, wird eine Laufzeitausnahme ausgelöst.

AWSSDK Pakete installieren mit NuGet

NuGetist ein Paketverwaltungssystem für die .NET-Plattform. Mit NuGet können Sie die AWSSDKPakete sowie mehrere andere Erweiterungen in Ihrem Projekt installieren. Weitere Informationen finden Sie im aws/dotnet-Repository auf der Website. GitHub

NuGet hat immer die neuesten Versionen der AWSSDK Pakete sowie frühere Versionen. NuGetist sich der Abhängigkeiten zwischen Paketen bewusst und installiert alle benötigten Pakete automatisch.

Warning

Die Liste der NuGet Pakete könnte eines enthalten, das einfach "AWSSDK" heißt (ohne angehängte Kennung). Installieren Sie dieses NuGet Paket NICHT. Es ist veraltet und sollte nicht für neue Projekte verwendet werden.

Pakete, NuGet die mit installiert wurden, werden zusammen mit Ihrem Projekt und nicht an einem zentralen Ort gespeichert. Auf diese Weise können Sie anwendungsspezifische Versionen von Komponenten installieren, ohne für andere Anwendungen Kompatibilitätsprobleme zu verursachen. Weitere Informationen zu NuGet finden Sie in der NuGet Dokumentation.

Note

Wenn Sie NuGet Pakete nicht pro Projekt herunterladen und installieren können oder dürfen, können Sie die AWSSDK Assemblys abrufen und lokal (oder lokal) speichern.
Falls dies auf Sie zutrifft und Sie die AWSSDK Assemblys noch nicht erhalten haben, finden Sie weitere Informationen unterBeschaffung von AWSSDK Baugruppen. Informationen zur Verwendung der lokal gespeicherten Assemblys finden Sie unterInstallieren Sie AWSSDK Baugruppen ohne NuGet.

Verwendung NuGet über die Befehlszeile oder das Terminal

1. Gehen Sie zu den AWSSDK Paketen auf NuGet und ermitteln Sie, welche Pakete Sie in Ihrem Projekt benötigen, z. B. AWSSDK.S3.
2. Kopieren Sie den .NET-CLI-Befehl von der Webseite dieses Pakets, wie im folgenden Beispiel gezeigt.

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

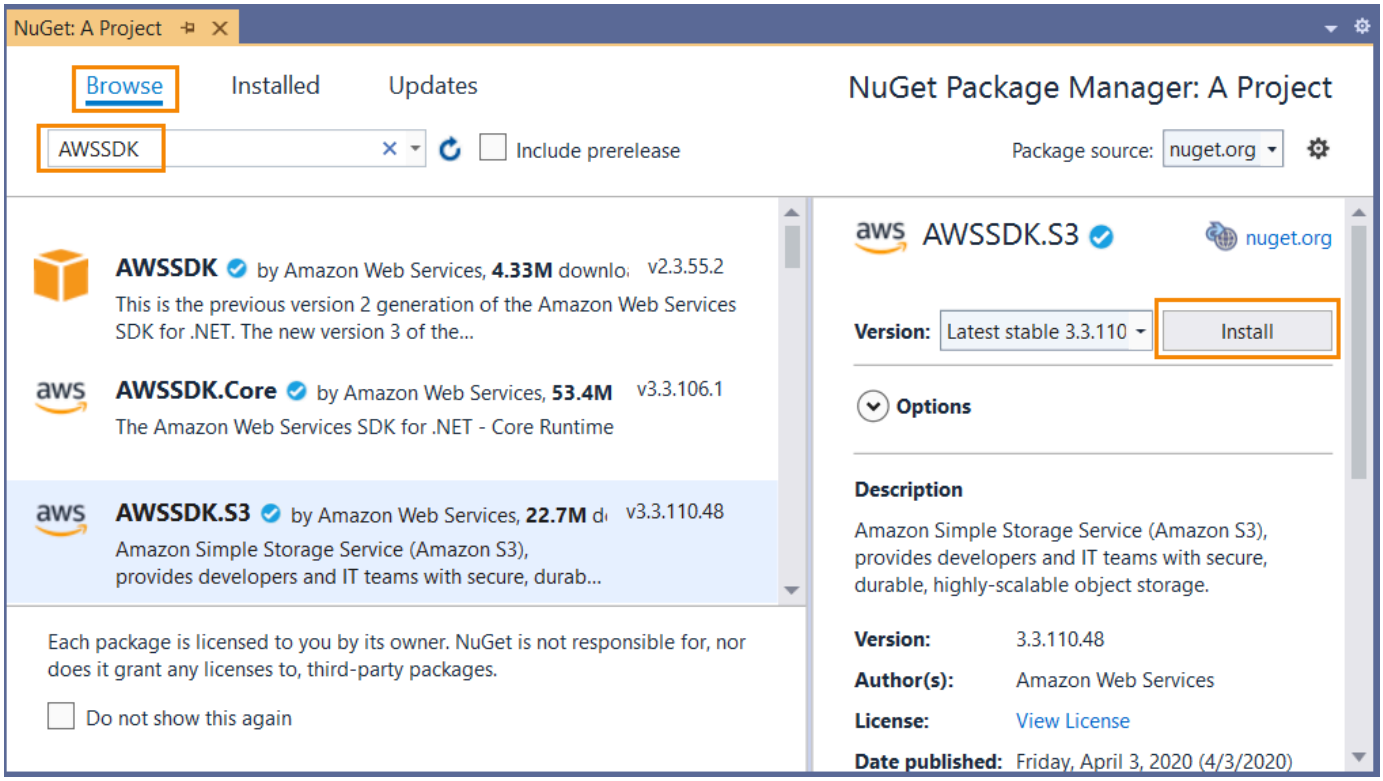
3. Führen Sie im Verzeichnis Ihres Projekts den.NET-CLI-Befehl aus. NuGet installiert auch alle Abhängigkeiten, z. B. AWSSDK.Core.

> **Note**
>
> Wenn Sie nur die neueste Version eines NuGet Pakets benötigen, können Sie Versionsinformationen aus dem Befehl ausschließen, wie im folgenden Beispiel gezeigt.
> `dotnet add package AWSSDK.S3`

Verwenden NuGet im Visual Studio Solution Explorer

1. Klicken Sie im Projektmappen-Explorer mit der rechten Maustaste auf Ihr Projekt und wählen Sie dann im Kontextmenü NuGet Pakete verwalten aus.
2. Wählen Sie im linken Bereich des NuGet Package Managers die Option Durchsuchen aus. Anschließend können Sie das Suchfeld verwenden, um nach dem Paket zu suchen, das Sie installieren möchten. NuGet installiert auch alle Abhängigkeiten, z. B. AWSSDK.Core.

   Die folgende Abbildung zeigt die Installation des AWSSDK.S3-Pakets.



NuGet Von der Package Manager Console aus verwenden

Wählen Sie in Visual Studio Tools, NuGet Package Manager, Package Manager Console aus.

Sie können die gewünschten AWSSDK Pakete von der Package Manager Console aus installieren, indem Sie den **`Install-Package`**Befehl verwenden. Verwenden Sie beispielsweise den folgenden Befehl, um AWSSDK.S3 zu installieren.

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

NuGet installiert auch alle Abhängigkeiten, z. B. AWSSDK.Core.

Wenn Sie eine frühere Version eines Pakets installieren müssen, verwenden Sie die `-Version` Option und geben Sie die gewünschte Paketversion an, wie im folgenden Beispiel gezeigt.

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

Weitere Informationen zu Befehlen der Package Manager Console finden Sie in der PowerShellReferenz in der NuGetDokumentation von Microsoft.

Installieren Sie AWSSDK Baugruppen ohne NuGet

In diesem Thema wird beschrieben, wie Sie die AWSSDK Assemblys verwenden können, die Sie lokal (oder lokal) abgerufen und gespeichert haben, wie unter beschriebenBeschaffung von AWSSDK Baugruppen. Dies ist nicht die empfohlene Methode für den Umgang mit SDK-Verweisen, ist jedoch in einigen Umgebungen erforderlich.

> **Note**
>
> Die empfohlene Methode für den Umgang mit SDK-Referenzen besteht darin, nur die NuGet Pakete herunterzuladen und zu installieren, die jedes Projekt benötigt. Diese Methode wird unter beschrieben AWSSDK Pakete installieren mit NuGet.

Um AWSSDK Baugruppen zu installieren

1. Erstellen Sie in Ihrem Projektbereich einen Ordner für die erforderlichen AWSSDK Assemblys. Als Beispiel könnten Sie diesen Ordner nennen`AwsAssemblies`.
2. Falls Sie dies noch nicht getan haben, besorgen Sie sich die AWSSDK Assemblys, wodurch die Assemblys in einem lokalen Download- oder Installationsordner abgelegt werden. Kopieren Sie die DLL-Dateien für die erforderlichen Assemblys aus diesem Download-Ordner in Ihr Projekt (in unserem Beispiel in den `AwsAssemblies` Ordner).

   Achten Sie darauf, auch alle Abhängigkeiten zu kopieren. Informationen zu Abhängigkeiten finden Sie auf der GitHubWebsite.
3. Verweisen Sie wie folgt auf die erforderlichen Baugruppen.

   Cross-platform development

   1. Öffnen Sie die `.csproj` Datei Ihres Projekts und fügen Sie ein `<ItemGroup>` Element hinzu.
   2. Fügen Sie dem `<ItemGroup>` Element ein `<Reference>` Element mit einem `Include` Attribut für jede erforderliche Assembly hinzu.

      Für Amazon S3 würden Sie beispielsweise die folgenden Zeilen zur `.csproj` Datei Ihres Projekts hinzufügen.

      Unter Linux und macOS:

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

Unter Windows:

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

3. Speichern Sie die `.csproj` Datei Ihres Projekts.

Windows with Visual Studio and .NET Core

1. Laden Sie in Visual Studio Ihr Projekt und öffnen Sie Project, Add Reference.
2. Wählen Sie unten im Dialogfeld die Schaltfläche „Durchsuchen“. Navigieren Sie zum Ordner Ihres Projekts und zu dem Unterordner, in den Sie die erforderlichen DLL-Dateien kopiert haben (`AwsAssembliesz`. B.).
3. Wählen Sie alle DLL-Dateien aus, klicken Sie auf Hinzufügen und anschließend auf OK.
4. Speichern Sie Ihr Projekt.

Festlegung der AWS Region für AWS SDK für .NET

AWS Regionen ermöglichen Ihnen den Zugriff auf AWS Dienste, die sich physisch in einer bestimmten geografischen Region befinden. Dies ist nicht nur für die Redundanz nützlich, sondern sorgt auch dafür, dass Ihre Daten und Anwendungen in der Nähe Ihres Standorts sowie des Standorts Ihrer Benutzer ausgeführt werden.

Eine aktuelle Liste aller unterstützten Regionen und Endpunkte für jeden AWS Dienst finden Sie unter Dienstendpunkte und Kontingente in der. Allgemeine AWS-Referenz Eine Liste der vorhandenen regionalen Endpunkte finden Sie unter AWS Dienstendpunkte. Ausführliche Informationen zu Regionen finden Sie unter Geben Sie an, welche AWS Regionen Ihr Konto verwenden kann.

Sie können einen AWS Service-Client für eine bestimmte Region erstellen. Sie können Ihre Anwendung auch mit einer Region konfigurieren, die für alle AWS Service-Clients verwendet wird. Diese beiden Fälle werden als Nächstes erklärt.

Erstellen Sie einen Service-Client mit einer bestimmten Region

Sie können die Region für jeden der AWS Service-Clients in Ihrer Anwendung angeben. Die Einstellung der Region auf diese Weise hat Vorrang vor allen globalen Einstellungen für diesen bestimmten Service-Client.

Bestehende Region

Dieses Beispiel zeigt Ihnen, wie Sie einen EC2 Amazon-Client in einer vorhandenen Region instanziieren. Es verwendet definierte RegionEndpointFelder.

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

Neue Region mit RegionEndpoint Klasse

Dieses Beispiel zeigt Ihnen, wie Sie mithilfe von einen neuen Regionen-Endpunkt erstellen RegionEndpoint. 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
}
```

Neue Region, die die Service-Client-Konfigurationsklasse verwendet

Dieses Beispiel zeigt Ihnen, wie Sie die `ServiceURL` Eigenschaft der Service-Client-Konfigurationsklasse verwenden, um die Region anzugeben; in diesem Fall mit der Amazon EC2 Config-Klasse.

Diese Technik funktioniert auch dann, wenn der Regions-Endpunkt nicht dem regulären Region-Endpunktmuster folgt.

```
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
}
```

Geben Sie eine Region für alle Service-Clients an

Es gibt mehrere Möglichkeiten, eine Region für alle AWS Service-Clients anzugeben, die Ihre Anwendung erstellt. Diese Region wird für Service-Clients verwendet, die nicht mit einer bestimmten Region erstellt wurden.

Der AWS SDK für .NET sucht in der folgenden Reihenfolge nach einem Regionswert.

Profile

Geben Sie ein Profil ein, das Ihre Anwendung oder das SDK geladen hat. Weitere Informationen finden Sie unter Auflösung von Anmeldeinformationen und Profilen.

Umgebungsvariablen

In der `AWS_REGION` Umgebungsvariablen festgelegt.

Unter Linux oder macOS:

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

Unter Windows:

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

> **Note**
>
> Wenn Sie diese Umgebungsvariable für das gesamte System festlegen (mit `export` oder`setx`), wirkt sie sich auf alle SDKs Toolkits aus, nicht nur auf die AWS SDK für .NET.

AWSConfigs Klasse

Als festgelegt AWSConfigs. AWSRegionEigentum.

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

Auflösung der Region

Wenn keine der oben beschriebenen Methoden zur Angabe von verwendet wird AWS-Region, wird AWS SDK für .NET versucht, eine Region zu finden, in der der AWS Service-Client arbeiten soll.

Reihenfolge der Auflösung der Region

1. Anwendungskonfigurationsdateien wie `app.config` und`web.config`.
2. Umgebungsvariablen (`AWS_REGION`und`AWS_DEFAULT_REGION`).
3. Ein Profil mit dem Namen, der durch einen Wert in angegeben wird`AWSConfigs.AWSProfileName`.
4. Ein Profil mit dem durch die `AWS_PROFILE` Umgebungsvariable angegebenen Namen.
5. Das `[default]` Profil.
6. EC2 Amazon-Instance-Metadaten (wenn sie auf einer EC2 Instance ausgeführt werden).

Wenn keine Region gefunden wird, löst das SDK eine Ausnahme aus, die besagt, dass der AWS Service-Client keine konfigurierte Region hat.

Besondere Informationen über die Region China (Peking)

Um Services in der Region China (Peking) verwenden zu können, benötigen Sie ein Konto und Anmeldeinformationen, die spezifisch für die Region China (Peking) sind. Konten und Anmeldeinformationen für andere AWS Regionen funktionieren nicht für die Region China (Peking). Ebenso funktionieren Konten und Anmeldeinformationen für die Region China (Peking) nicht für andere AWS Regionen. Informationen zu Endpunkten und Protokollen, die in der Region China (Peking) verfügbar sind, finden Sie unter Endpunkte in der Region Peking.

Besondere Informationen zu neuen Diensten AWS

Neue AWS Dienste können zunächst in einigen Regionen eingeführt und dann in anderen Regionen unterstützt werden. In diesen Fällen müssen Sie nicht das neueste SDK installieren, um für diesen Dienst auf die neuen Regionen zuzugreifen. Sie können neu hinzugefügte Regionen auf Kundenbasis oder global angeben, wie bereits gezeigt.

Auflösung von Anmeldeinformationen und Profilen

Der AWS SDK für .NET sucht in einer bestimmten Reihenfolge nach Anmeldeinformationen und verwendet den ersten verfügbaren Satz für die aktuelle Anwendung.

Reihenfolge bei der Suche nach Anmeldeinformationen

1. Anmeldeinformationen, die explizit auf dem AWS Service-Client festgelegt wurden, wie unter beschriebenZugreifen auf Anmeldeinformationen und Profile in einer Anwendung.

   > **Note**
   >
   > Dieses Thema ist in diesem Besondere Überlegungen Abschnitt enthalten, da es nicht die bevorzugte Methode zur Angabe von Anmeldeinformationen ist.

2. Sitzungen AWSCredentials, die aus den `AWS_SESSION_TOKEN` Umgebungsvariablen `AWS_ACCESS_KEY_IDAWS_SECRET_ACCESS_KEY`, und erstellt werden, wenn alle drei Variablen einen Wert haben.
3. Basic AWSCredentials, die aus den Umgebungsvariablen `AWS_ACCESS_KEY_ID` und `AWS_SECRET_ACCESS_KEY` Umgebungsvariablen erstellt werden, wenn beide Variablen einen Wert haben.
4. AssumeRoleWithWebIdentityCredentialsdie aus den `AWS_ROLE_ARN` Umgebungsvariablen `AWS_WEB_IDENTITY_TOKEN_FILE` und erzeugt werden, wenn beide Variablen einen Wert haben.
5. Ein Anmeldeinformationsprofil mit dem Namen, der durch einen Wert in angegeben wird AWSConfigs. AWSProfileName.
6. Ein Anmeldeinformationsprofil mit dem in der `AWS_PROFILE` Umgebungsvariablen angegebenen Namen.
7. Das `[default]`-Anmeldeinformationsprofil.

8. Der Anbieter für Container-Anmeldeinformationen.
9. Metadaten der EC2 Amazon-Instanz.

Wenn Ihre Anwendung auf einer EC2 Amazon-Instance ausgeführt wird, z. B. in einer Produktionsumgebung, verwenden Sie eine IAM-Rolle, wie unter beschriebenZugriff mithilfe einer IAM-Rolle gewähren. Andernfalls, z. B. bei Tests vor der Veröffentlichung, speichern Sie Ihre Anmeldeinformationen in einer Datei, die das Dateiformat für AWS Anmeldeinformationen verwendet, auf das Ihre Webanwendung auf dem Server Zugriff hat.

Weitere Informationen zu Umgebungsvariablen und Profilen mit Anmeldeinformationen finden Sie in den folgenden Themen im Referenzhandbuch AWS SDKs und im Tools-Referenzhandbuch: Umgebungsvariablen, Liste der Umgebungsvariablen und Dateien mit gemeinsam genutzten Konfigurationen und Anmeldeinformationen.

Auflösung des Profils

Da es zwei verschiedene Speichermechanismen für Anmeldeinformationen gibt, ist es wichtig zu verstehen, wie diese konfiguriert werden müssen AWS SDK für .NET , um sie zu verwenden. Das AWSConfigs. AWSProfilesDie Standort-Eigenschaft steuert, wie die Anmeldeinformationsprofile AWS SDK für .NET findet.

AWSProfilesSpeicherort	Verhalten bei der Profilauflösung
Null (nicht festgelegt) oder leer	Suchen Sie im SDK-Store, falls die Plattform dies unterstützt, und suchen Sie dann im Standardverzeichnis nach der Datei mit den gemeinsamen AWS Anmeldeinforma tionen. Wenn sich das Profil an keinem dieser Speicherorte befindet, suchen Sie `~/.aws/co nfig` (Linux oder macOS) oder `%USERPROF ILE%\.aws\config` (Windows).
Der Pfad zu einer Datei im Format der AWS Anmeldeinformationsdatei	Suchen Sie zunächst nur die angegebene Datei für ein Profil mit dem angegebenen Namen.

Verwenden von Anmeldeinformationen für Verbundbenutzerkonten

Anwendungen, die die AWS SDK für .NET (AWSSDK.Core-Version 3.1.6.0 und höher) verwenden, können Verbundbenutzerkonten über Active Directory Federation Services (AD FS) verwenden, um mithilfe von Security Assertion Markup Language (SAML) auf AWS Dienste zuzugreifen.

Verbundener Zugriffsunterstützung bedeutet, dass sich Benutzer mithilfe Ihrer Active Directory authentifizieren können. Temporäre Anmeldeinformationen werden dem Benutzer automatisch erteilt. Diese temporären Anmeldeinformationen, die eine Stunde lang gültig sind, werden verwendet, wenn Ihre Anwendung Dienste aufruft. AWS Das SDK übernimmt die Verwaltung der temporären Anmeldeinformationen. Wenn Ihre Anwendung bei Benutzerkonten, die mit einer Domäne verbunden sind, einen Aufruf durchführt, die Anmeldeinformationen jedoch abgelaufen sind, wird der Benutzer automatisch neu authentifiziert, und es werden neue Anmeldeinformationen erteilt. (Bei Konten, die keiner Domäne angehören, wird der Benutzer vor der erneuten Authentifizierung aufgefordert, Anmeldeinformationen einzugeben.)

Um diese Unterstützung in Ihrer .NET-Anwendung nutzen zu können, müssen Sie zunächst das Rollenprofil mithilfe eines PowerShell Cmdlets einrichten. Informationen dazu finden Sie in der AWS Tools for Windows PowerShell Dokumentation.

Nachdem Sie das Rollenprofil eingerichtet haben, verweisen Sie in Ihrer Anwendung auf das Profil. Es gibt eine Reihe von Möglichkeiten, dies zu tun. Eine davon ist die Verwendung von AWSConfigs. AWSProfileBenennen Sie die Eigenschaft genauso, wie Sie es mit anderen Anmeldeinformationsprofilen tun würden.

Die AWS Security Token ServiceAssembly (AWSSDK. SecurityToken) bietet die SAML-Unterstützung zum Abrufen von AWS Anmeldeinformationen. Wenn Sie Anmeldeinformationen für Verbundbenutzerkonten verwenden möchten, stellen Sie sicher, dass diese Assembly für Ihre Anwendung verfügbar ist.

Angeben von Rollen oder temporären Anmeldeinformationen

Für Anwendungen, die auf EC2 Amazon-Instances ausgeführt werden, ist die sicherste Methode zur Verwaltung von Anmeldeinformationen die Verwendung von IAM-Rollen, wie unter beschriebenZugriff mithilfe einer IAM-Rolle gewähren.

Für Anwendungsszenarien, in denen die ausführbare Software Benutzern außerhalb Ihrer Organisation zur Verfügung steht, empfehlen wir, dass Sie die Software so entwerfen, dass temporäre Sicherheitsanmeldedaten verwendet werden. Diese Anmeldeinformationen bieten nicht

nur eingeschränkten Zugriff auf AWS Ressourcen, sondern haben auch den Vorteil, dass sie nach einem bestimmten Zeitraum ablaufen. Weitere Informationen zum Verwenden von temporären Sicherheitsanmeldeinformationen finden Sie in den folgenden Themen:

- Temporäre Sicherheitsnachweise
- Amazon Cognito Cognito-Identitätspools

Verwendung von Proxy-Anmeldeinformationen

Wenn Ihre Software mit AWS über einen Proxy kommuniziert, können Sie Anmeldeinformationen für den Proxy angeben, indem Sie die `ProxyCredentials` Eigenschaft der `Config` Klasse eines Dienstes verwenden. Die `Config` Klasse eines Dienstes ist normalerweise Teil des primären Namespaces für den Dienst. Zu den Beispielen gehören die folgenden: AmazonCloudDirectoryConfigim Amazon. CloudDirectoryNamespace und AmazonGameLiftConfigim Amazon. GameLiftNamespace.

Für Amazon S3 könnten Sie beispielsweise Code verwenden, der dem folgenden ähnelt, wobei `SecurelyStoredUserName` und für den Proxy-Benutzernamen und das Passwort `SecurelyStoredPassword` stehen, die in einem NetworkCredentialObjekt angegeben sind.

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

> **Note**
>
> In früheren Versionen von SDK wurde `ProxyUsername` und `ProxyPassword` verwendet, diese Eigenschaften sind jedoch veraltet.

Wiederholungen und Timeouts

AWS SDK für .NET Damit können Sie die Anzahl der Wiederholungen und die Timeout-Werte für HTTP-Anfragen an Dienste konfigurieren. AWS Wenn die Standardwerte für Wiederholversuche und Zeitüberschreitungen für Ihre Anwendung nicht geeignet sind, können Sie sie an Ihre speziellen Anforderungen anpassen. Es ist jedoch wichtig, zu verstehen, wie sich dies auf das Verhalten Ihrer Anwendung auswirkt.

Bei der Bestimmung der Werte für Wiederholversuche und Zeitüberschreitungen gilt es, folgende Überlegungen anzustellen:

- Wie sollen die AWS SDK für .NET und Ihre Anwendung reagieren, wenn sich die Netzwerkkonnektivität verschlechtert oder ein AWS Dienst nicht erreichbar ist? Soll der Aufruf schnell fehlschlagen oder ist es angemessen, wenn er in Ihrem Auftrag wiederholt versucht wird?
- Handelt es sich bei Ihrer Anwendung um eine benutzerbezogene Anwendung bzw. Website, die reaktionsfähig sein muss, oder ist es eine Hintergrundverarbeitung mit einer höheren Toleranz für mehr Latenzen?
- Wird die Anwendung in einem zuverlässigen Netzwerk mit geringer Latenz oder an einem Remote-Standort mit unzuverlässiger Konnektivität bereitgestellt?

Wiederholversuche

Übersicht

Sie AWS SDK für .NET können Anfragen wiederholen, die aufgrund serverseitiger Drosselung oder Verbindungsabbrüche fehlschlagen. Es gibt zwei Eigenschaften von Dienstkonfigurationsklassen, mit denen Sie das Wiederholungsverhalten eines Dienstclients angeben können. Servicekonfigurationsklassen erben diese Eigenschaften von der abstrakten Amazon.Runtime. ClientConfigKlasse der API-Referenz:AWS SDK für .NET

- `RetryMode`gibt einen von drei Wiederholungsmodi an, die in der Amazon.Runtime definiert sind. RequestRetryModeAufzählung.

  Der Standardwert für Ihre Anwendung kann mithilfe der `AWS_RETRY_MODE` Umgebungsvariablen oder der Einstellung retry_mode in der gemeinsam genutzten Konfigurationsdatei gesteuert werden. AWS

- `MaxErrorRetry`gibt die Anzahl der Wiederholungen an, die auf Service-Client-Ebene zulässig sind. Das SDK wiederholt den Vorgang so oft, bis ein Fehler auftritt und eine Ausnahme ausgelöst wird.

  Der Standardwert für Ihre Anwendung kann mithilfe der `AWS_MAX_ATTEMPTS` Umgebungsvariablen oder der Einstellung max_attempts in der gemeinsam genutzten Konfigurationsdatei gesteuert werden. AWS

Detaillierte Beschreibungen dieser Eigenschaften finden Sie in der Zusammenfassung Amazon.Runtime. ClientConfigKlasse der AWS SDK für .NET API-Referenz. Jeder Wert von `RetryMode` entspricht standardmäßig einem bestimmten Wert von`MaxErrorRetry`, wie in der folgenden Tabelle dargestellt.

RetryMode	Entsprechend MaxErrorRetry (Amazon DynamoDB)	Entspricht MaxErrorRetry (alle anderen)
Veraltet	10	4
Standard	10	2
Adaptiv (experimentell)	10	2

Behavior

Wenn Ihre Anwendung gestartet wird

Wenn Ihre Anwendung gestartet wird, `MaxErrorRetry` werden Standardwerte für das SDK konfiguriert `RetryMode` und vom SDK konfiguriert. Diese Standardwerte werden verwendet, wenn Sie einen Service-Client erstellen, sofern Sie keine anderen Werte angeben.

- Wenn die Eigenschaften in Ihrer Umgebung nicht festgelegt sind, `RetryMode` ist der Standard für als Legacy konfiguriert und der Standard für `MaxErrorRetry` wird mit dem entsprechenden Wert aus der vorherigen Tabelle konfiguriert.
- Wenn der Wiederholungsmodus in Ihrer Umgebung festgelegt wurde, wird dieser Wert als Standard für `RetryMode` verwendet. Der Standardwert für `MaxErrorRetry` ist mit dem entsprechenden Wert aus der vorherigen Tabelle konfiguriert, es sei denn, der Wert für die maximale Anzahl von Fehlern wurde auch in Ihrer Umgebung festgelegt (siehe unten).
- Wenn der Wert für die maximale Anzahl von Fehlern in Ihrer Umgebung festgelegt wurde, wird dieser Wert als Standard für verwendet`MaxErrorRetry`. Amazon DynamoDB ist die Ausnahme von dieser Regel. Der DynamoDB-Standardwert für `MaxErrorRetry` ist immer der Wert aus der vorherigen Tabelle.

Während Ihre Anwendung ausgeführt wird

Wenn Sie einen Service-Client erstellen, können Sie die Standardwerte für `RetryMode` und verwenden`MaxErrorRetry`, wie bereits beschrieben, oder Sie können andere Werte angeben. Um andere Werte anzugeben, erstellen Sie ein Servicekonfigurationsobjekt wie AmazonDynamoDBConfigAmazon und fügen Sie es SQSConfig hinzu, wenn Sie den Service-Client erstellen.

Diese Werte können für einen Service-Client nicht mehr geändert werden, nachdem er erstellt wurde.

Überlegungen

Wenn ein erneuter Versuch erfolgt, erhöht sich die Latenz Ihrer Anfrage. Sie sollten Ihre Wiederholversuche basierend auf den Anwendungsbeschränkungen für die Gesamtanfragenlatenz und die Fehlerraten konfigurieren.

Timeouts

Auf diese AWS SDK für .NET Weise können Sie Timeouts für Anfragen auf Service-Client-Ebene und pro Methodenaufruf konfigurieren. Es gibt zwei Mechanismen für die Konfiguration von Timeouts, die in den folgenden Abschnitten behandelt werden:

- Wenn Sie asynchrone Aufrufe verwenden, können Sie den `CancellationToken` Parameter der Methode verwenden.
- Wenn Sie synchrone Aufrufe in.NET Framework verwenden, können Sie die `ReadWriteTimeout` Eigenschaften `Timeout` und der abstrakten Amazon.Runtime verwenden. ClientConfigKlasse.

Den **`CancellationToken`** Parameter für Timeouts verwenden

AWS SDK für .NET Mit dem können Sie mithilfe des Parameters Zeitüberschreitungen für Anfragen bei asynchronen Aufrufen konfigurieren. `CancellationToken` Der folgende Codeausschnitt zeigt ein Beispiel. Der Code wird ausgelöst, `System.Threading.Tasks.TaskCanceledException` wenn die Anfrage nicht innerhalb von 10 Sekunden abgeschlossen ist.

```
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);
}
```

Verwenden der **`ReadWriteTimeout`** Eigenschaften **`Timeout`** und für Timeouts

> **Note**
>
> Die `Timeout` Eigenschaft hat keine Auswirkungen auf asynchrone Aufrufe. Wenn Sie asynchrone Aufrufe verwenden, finden Den `CancellationToken` Parameter für Timeouts verwenden Sie stattdessen weitere Informationen.

AWS SDK für .NET Damit können Sie die Werte für das Anforderungs-Timeout und das Socket-Timeout für Lese-/Schreibvorgänge auf Service-Client-Ebene konfigurieren. Diese Werte sind in der `Timeout` und den `ReadWriteTimeout` Eigenschaften der abstrakten Amazon.Runtime angegeben. ClientConfigKlasse. Diese Werte werden als `ReadWriteTimeout` Eigenschaften `Timeout` und der HttpWebRequestObjekte weitergegeben, die vom AWS Service-Client-Objekt erstellt wurden. Standardmäßig beträgt der `Timeout`-Wert 100 Sekunden und der `ReadWriteTimeout`-Wert 300 Sekunden.

Wenn Ihr Netzwerk eine hohe Latenz aufweist oder Bedingungen vorliegen, die bei einer Operation einen Wiederholversuch auslösen, kann die Verwendung der Werte für eine lange Zeitüberschreitung und einer hohen Anzahl von Wiederholversuchen dazu führen, dass einige SDK-Operationen scheinbar nicht mehr reagieren.

> **Note**
>
> Die Version von AWS SDK für .NET , die auf die Portable Class Library (PCL) abzielt, verwendet die HttpClientKlasse anstelle der `HttpWebRequest` Klasse und unterstützt nur die Timeout-Eigenschaft.

Nachstehend sind die Ausnahmen von den Standardwerten für Zeitüberschreitungen angeführt. Diese Werte werden überschrieben, wenn Sie die Werte für die Zeitüberschreitung explizit angeben.

- `Timeout`und `ReadWriteTimeout` werden auf die Maximalwerte gesetzt, wenn die aufgerufene Methode einen Stream hochlädt, z. B. AmazonS3Client. PutObjectAsync(), AmazonS3Client. UploadPartAsync(),. AmazonGlacierClient UploadArchiveAsync() und so weiter.
- Die Versionen des AWS SDK für .NET Ziel-.NET Frameworks haben die Maximalwerte für alle AmazonS3Client `Timeout` und `ReadWriteTimeout` Objekte festgelegt. AmazonGlacierClient
- Die Versionen von AWS SDK für .NET , die auf die Portable Class Library (PCL) und .NET Core abzielen, sind `Timeout` auf den Maximalwert für alle AmazonS3Client und Objekte eingestellt. AmazonGlacierClient

Das folgende Beispiel zeigt Ihnen, wie Sie den Standard-Wiederholungsmodus, maximal 3 Wiederholungen, ein Timeout von 10 Sekunden und ein Lese-/Schreib-Timeout von 10 Sekunden (falls zutreffend) angeben. Dem Amazons3Client-Konstruktor wird ein AmazonS3Config-Objekt zugewiesen.

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

Beobachtbarkeit

Beobachtbarkeit ist das Ausmaß, in dem der aktuelle Zustand eines Systems aus den ausgegebenen Daten abgeleitet werden kann. Die ausgegebenen Daten werden allgemein als Telemetrie bezeichnet.

AWS SDK für .NET Sie können zwei gängige Telemetriesignale, Metriken und Traces, sowie Protokollierung bereitstellen. Sie können eine Verbindung herstellen TelemetryProvider, um Telemetriedaten an ein Observability-Backend (z. B. AWS X-Rayoder Amazon CloudWatch) zu senden und dann darauf zu reagieren.

Standardmäßig sind Telemetriesignale im SDK deaktiviert. In diesem Thema wird erklärt, wie die Telemetrieausgabe aktiviert und konfiguriert wird.

Weitere Ressourcen

Weitere Informationen zur Aktivierung und Verwendung von Observability finden Sie in den folgenden Ressourcen:

- OpenTelemetry
- Der Blogbeitrag Enhancing Observability in the AWS SDK für .NET with. OpenTelemetry
- Der Blogbeitrag Ankündigung der allgemeinen Verfügbarkeit AWS OpenTelemetry von.NET-Bibliotheken.
- Exporteure für OpenTelemetry
- Beispiele für Observability in finden Sie unter Observability im Tools for PowerShell User Guide. AWS -Tools für PowerShell

Konfigurieren Sie ein `TelemetryProvider`

Sie können a `TelemetryProvider` in Ihrer Anwendung global für alle Service-Clients oder für einzelne Clients konfigurieren, wie in den folgenden Beispielen gezeigt. Der the section called "Telemetrieanbieter" Abschnitt enthält Informationen zu Telemetrie-Implementierungen, einschließlich Informationen zu Implementierungen, die mit dem SDK bereitgestellt werden.

Konfigurieren Sie den globalen Standardtelemetrie-Anbieter

Standardmäßig versucht jeder Dienstclient, den weltweit verfügbaren Telemetrieanbieter zu verwenden. Auf diese Weise können Sie den Anbieter einmal festlegen, und alle Clients verwenden ihn. Dies sollte nur einmal durchgeführt werden, bevor Sie Service-Clients erstellen.

Der folgende Codeausschnitt zeigt Ihnen, wie Sie den globalen Telemetrieanbieter einrichten. Anschließend wird ein Amazon S3 S3-Serviceclient erstellt und versucht, einen fehlgeschlagenen Vorgang auszuführen. Der Code fügt der Anwendung sowohl Tracing als auch Metriken hinzu. Dieser Code verwendet die folgenden NuGet Pakete: `OpenTelemetry.Exporter.Console` und`OpenTelemetry.Instrumentation.AWS`.

> **Note**
>
> Wenn Sie AWS IAM Identity Center für die Authentifizierung verwenden, achten Sie darauf, auch `AWSSDK.SSO` und hinzuzufügen`AWSSDK.SSOOIDC`.

```
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();
```

Konfigurieren Sie einen Telemetrieanbieter für einen bestimmten Service-Client

Sie können einen einzelnen Dienstclient mit einem bestimmten Telemetrieanbieter (mit Ausnahme des globalen) konfigurieren. Verwenden Sie dazu die `TelemetryProvider` Klasse des Config-Objekts eines Service-Client-Konstruktors. Sehen Sie sich zum Beispiel AmazonS3Config an und suchen Sie nach der Eigenschaft. `TelemetryProvider` Informationen zu benutzerdefinierten the section called “Telemetrieanbieter” Telemetrie-Implementierungen finden Sie unter.

Themen

- Metriken
- Telemetrieanbieter

Metriken

In der folgenden Tabelle sind die Telemetriemetriken aufgeführt, die das SDK ausgibt. Konfigurieren Sie einen Telemetrieanbieter, um die Messwerte beobachtbar zu machen.

Welche Metriken werden ausgegeben?

Metrikname	Einheiten	Typ	Attribute	Beschreibung
client.call.duration	S	Histogram	rpc.service, rpc.method	Gesamtdauer des Anrufs (einschließlich Wiederholungsversuche und Zeit zum Senden oder Empfangen von Anfrage und Antworttext)
client.uptime	S	Histogram	rpc.service	Die Zeit, seit ein Client erstellt wurde
client.call.attempts	{Versuch}	Monotonic Counter	rpc.service, rpc.method	Die Anzahl der Versuche für einen einzelnen Vorgang
client.call.errors	{Fehler}	Monotonic Counter	rpc.service, rpc.method, exception.type	Die Anzahl der Fehler bei einem Vorgang
client.call.attempt_duration	S	Histogram	rpc.service, rpc.method	Die Zeit, die benötigt wird, um eine Verbindung mit dem Dienst herzustellen, die Anfrage zu senden und den

Metrikname	Einheiten	Typ	Attribute	Beschreibung
				HTTP-Statuscode und die Header zurückzubekommen (einschließlich der Wartezeit in der Warteschlange auf das Senden)
client.call.resolve_endpoint_duration	S	Histogram	rpc.service, rpc.method	Die Zeit, die benötigt wird, um einen Endpunkt (Endpoint Resolver, nicht DNS) für die Anfrage aufzulösen
client.call.serialization_duration	S	Histogram	rpc.service, rpc.method	Die Zeit, die benötigt wird, um einen Nachrichtentext zu serialisieren
client.call.deserialization_duration	S	Histogram	rpc.service, rpc.method	Die Zeit, die benötigt wird, um einen Nachrichtentext zu deserialisieren
client.call.auth.signing_duration	S	Histogram	rpc.service, rpc.method	Die Zeit, die benötigt wird, um eine Anfrage zu signieren
client.call.auth.resolve_identity_duration	S	Histogram	rpc.service, rpc.method	Die Zeit, die benötigt wird, um eine Identität (wie AWS Anmeldeinformationen oder ein Inhabertoken) von einem Identitätsanbieter abzurufen
client.http.bytes_sent	Von	Monoton Counter	server.adresse	Die Gesamtzahl der vom HTTP-Client gesendeten Byte
client.http.bytes_received	Von	Monoton Counter	server.adresse	Die Gesamtzahl der vom HTTP-Client empfangenen Byte

Im Folgenden finden Sie die Spaltenbeschreibungen:

- Metrikname — Der Name der ausgegebenen Metrik.
- Einheiten — Die Maßeinheit für die Metrik. Die Einheiten werden in der UCUM-Notation mit Berücksichtigung der Groß- und Kleinschreibung („c/s“) angegeben.

- Typ — Die Art des Instruments, das zur Erfassung der Metrik verwendet wird.
- Attribute — Der Satz von Attributen (Dimensionen), die mit der Metrik ausgegeben werden.
- Beschreibung — Eine Beschreibung dessen, was die Metrik misst.

Telemetrieanbieter

Das SDK bietet eine Implementierung von OpenTelemetryals Telemetrieanbieter, die im nächsten Abschnitt beschrieben wird.

Wenn Sie spezielle Telemetrieanforderungen haben, bereits eine Telemetrielösung im Sinn haben oder eine genaue Kontrolle darüber benötigen, wie Telemetriedaten erfasst und verarbeitet werden, können Sie auch Ihren eigenen Telemetrieanbieter implementieren.

Registrieren Sie Ihre eigene Implementierung bei der Klasse. TelemetryProvider Das Folgende ist ein einfaches Beispiel dafür, wie Sie Ihr eigenes `TracerProvider` und registrieren können`MeterProvider`.

```
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());
```

Themen

- Konfigurieren Sie den OpenTelemetry basierten Telemetrieanbieter

Konfigurieren Sie den OpenTelemetry basierten Telemetrieanbieter

Das AWS SDK für .NET beinhaltet eine Implementierung eines OpenTelemetry basierten Telemetrieanbieters. Einzelheiten dazu, wie Sie diesen Anbieter als globalen Telemetrieanbieter einrichten, finden Sie unter. Konfigurieren Sie ein `TelemetryProvider` Um diesen Telemetrieanbieter verwenden zu können, benötigen Sie die folgenden Ressourcen in Ihrem Projekt:

- Das OpenTelemetry.instrumentation.aws-Paket NuGet .
- Ein Telemetrieexporteur wie OTLP oder Console. Weitere Informationen finden Sie in der Dokumentation unter Exporter. OpenTelemetry

Die im SDK enthaltene OpenTelemetry Implementierung kann so konfiguriert werden, dass weniger Ablaufverfolgung für HTTPS-Anfragen, Anmeldeinformationen und Komprimierung erforderlich ist. Stellen Sie dazu die `SuppressDownstreamInstrumentation` Option auf ein`true`, ähnlich wie im Folgenden:

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

Weitere Informationen zu diesem Anbieter finden Sie im Blogbeitrag Enhancing Observability in the AWS SDK für .NET with OpenTelemetry.

Zusätzliche Informationen über Benutzer und Rollen

Für die Entwicklung von.NET-Anwendungen AWS oder für die Ausführung AWS von.NET-Anwendungen benötigen Sie eine Kombination aus Benutzern, Berechtigungssätzen und Dienstrollen, die für diese Aufgaben geeignet sind.

Die spezifischen Benutzer, Berechtigungssätze und Servicerollen, die Sie erstellen, und die Art und Weise, wie Sie sie verwenden, hängen von den Anforderungen Ihrer Anwendungen ab. Im Folgenden finden Sie einige zusätzliche Informationen darüber, warum sie verwendet werden können und wie sie erstellt werden.

Benutzer und Berechtigungssätze

Es ist zwar möglich, ein IAM-Benutzerkonto mit langfristigen Anmeldeinformationen für den Zugriff auf AWS -Services zu verwenden, dies ist jedoch keine bewährte Methode mehr und sollte vermieden werden. Selbst während der Entwicklung hat es sich bewährt, Benutzer und Berechtigungssätze in einer Identitätsquelle zu erstellen AWS IAM Identity Center und temporäre Anmeldeinformationen zu verwenden, die von einer Identitätsquelle bereitgestellt werden.

Für die Entwicklung können Sie den Benutzer verwenden, den Sie erstellt haben oder den Sie in Authentifizierung mit AWS erhalten haben. Wenn Sie über die entsprechenden AWS Management Console Berechtigungen verfügen, können Sie auch verschiedene Berechtigungssätze mit den geringsten Rechten für diesen Benutzer erstellen oder neue Benutzer speziell für Entwicklungsprojekte erstellen, indem Sie Berechtigungssätze mit den geringsten Rechten bereitstellen. Die Vorgehensweise, die Sie auswählen (sofern Sie dies tun), hängt von Ihren Umständen ab.

Weitere Informationen zu diesen Benutzern und Berechtigungssätzen sowie zu deren Erstellung finden Sie unter Authentifizierung und Zugriff im AWS SDKs Referenzhandbuch zu Tools und Erste Schritte im AWS IAM Identity Center Benutzerhandbuch.

Servicerollen

Sie können eine AWS Servicerolle einrichten, um im Namen von Benutzern auf AWS Dienste zuzugreifen. Diese Art des Zugriffs ist geeignet, wenn mehrere Personen Ihre Anwendung remote ausführen, z. B. auf einer EC2 Amazon-Instance, die Sie zu diesem Zweck erstellt haben.

Das Verfahren zur Erstellung einer Servicerolle ist je nach Situation unterschiedlich, sieht aber im Wesentlichen wie folgt aus.

1. Melden Sie sich bei der an AWS Management Console und öffnen Sie die IAM-Konsole unter https://console.aws.amazon.com/iam/.
2. Wählen Sie Roles (Rollen) und anschließend Create role (Rolle erstellen).
3. Wählen Sie AWS Service, suchen und wählen Sie EC2(zum Beispiel) und wählen Sie dann den EC2Anwendungsfall aus (zum Beispiel).
4. Wählen Sie Weiter: Berechtigungen und wählen Sie die entsprechenden Richtlinien für die AWS Dienste aus, die Ihre Anwendung verwenden soll.

> **Warning**
>
> Wählen Sie NICHT die AdministratorAccessRichtlinie aus, da diese Richtlinie Lese- und Schreibberechtigungen für fast alles in Ihrem Konto ermöglicht.

5. Wählen Sie Weiter: Stichwörter und geben Sie die gewünschten Stichwörter ein.

   Informationen zu Tags finden Sie unter Steuern des Zugriffs mithilfe von AWS Ressourcen-Tags im IAM-Benutzerhandbuch.

6. Wählen Sie Weiter: Überprüfen und geben Sie einen Rollennamen und eine Rollenbeschreibung ein. Wählen Sie dann Create Role.

Allgemeine Informationen zu IAM-Rollen finden Sie unter Identitäten (Benutzer, Gruppen und Rollen) im IAM-Benutzerhandbuch. Ausführliche Informationen zu Rollen finden Sie im Thema IAM-Rollen in diesem Handbuch.

Zusätzliche Informationen zu Rollen

- Verwenden Sie IAM-Rollen für Aufgaben in Verbindung mit Aufgaben von Amazon Elastic Container Service (Amazon ECS).
- Verwenden Sie IAM-Rollen für Anwendungen, die auf EC2 Amazon-Instances ausgeführt werden.

Erweiterte Konfiguration für Ihr AWS SDK für .NET Projekt

Die Themen in diesem Abschnitt enthalten Informationen zu zusätzlichen Konfigurationsaufgaben und -methoden, die für Sie von Interesse sein könnten.

Themen

- AWSSDK.Extensions verwenden. NETCore.Setup und das Interface IConfiguration
- Konfigurieren von anderen Anwendungsparametern
- Referenz zu Konfigurationsdateien für AWS SDK für .NET

AWSSDK.Extensions verwenden. NETCore.Setup und das Interface IConfiguration

(Dieses Thema trug früher den Titel „Konfiguration AWS SDK für .NET mit .NET Core“)

Eine der größten Änderungen in.NET Core ist die Entfernung des Standards `ConfigurationManager` und der `web.config` Dateien, die mit.NET Framework `app.config` - und ASP.NET-Anwendungen verwendet wurden.

Die Konfiguration in.NET Core basiert auf Schlüssel-Wert-Paaren, die von Konfigurationsanbietern eingerichtet wurden. Konfigurationsanbieter lesen Konfigurationsdaten in Schlüssel-Wert-Paaren aus einer Vielzahl von Konfigurationsquellen, einschließlich Befehlszeilenargumenten, Verzeichnisdateien, Umgebungsvariablen und Einstellungsdateien.

> **Note**
>
> Weitere Informationen finden Sie unter Konfiguration in ASP.NET Core.

Um die Verwendung AWS SDK für .NET mit .NET Core zu vereinfachen, können Sie die .Extensions verwenden. AWSSDK NETCore.Setup-Paket NuGet . Wie viele andere .NET-Core-Bibliotheken fügt es der `IConfiguration` Schnittstelle Erweiterungsmethoden hinzu, um eine reibungslose AWS Konfiguration zu gewährleisten.

Der Quellcode für dieses Paket befindet sich GitHub unterhttps://github.com/aws/aws-sdk-net/tree/main/extensions/src/AWSSDK.Extensions.NETCore.Setup.

Verwenden von AWSSDK .Extensions. NETCore. Einrichtung

Angenommen, Sie erstellen eine ASP.NET Core-Anwendung Model-View-Controller (MVC), was mit der ASP.NET Core-Webanwendungsvorlage in Visual Studio oder durch Ausführung `dotnet new mvc ...` in der.NET Core-CLI erreicht werden kann. Wenn Sie eine solche Anwendung erstellen, `Startup.cs` verarbeitet der Konstruktor für die Konfiguration die Konfiguration, indem er verschiedene Eingabequellen von Konfigurationsanbietern einliest, wie z. `appsettings.json`

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

Um das `Configuration` Objekt zum Abrufen der AWSOptionen zu verwenden, fügen Sie zuerst das `AWSSDK.Extensions.NETCore.Setup` NuGet Paket hinzu. Fügen Sie dann Ihre Optionen wie im Folgenden beschrieben zur Konfigurationsdatei hinzu.

Beachten Sie, dass eine der zu Ihrem Projekt hinzugefügten Dateien `appsettings.Development.json` Dies entspricht einem `EnvironmentName` Satz von Development. Während der Entwicklung fügen Sie Ihre Konfiguration in diese Datei ein, die nur bei lokalen Tests gelesen wird. Wenn Sie eine EC2 Amazon-Instance bereitstellen, die auf Production **`EnvironmentName`** eingestellt ist, wird diese Datei ignoriert und es AWS SDK für .NET werden die IAM-Anmeldeinformationen und die Region verwendet, die für die EC2 Amazon-Instance konfiguriert sind.

Die folgenden Konfigurationseinstellungen zeigen Beispiele für die Werte, die Sie der `appsettings.Development.json` Datei in Ihrem Projekt hinzufügen können, um AWS Einstellungen bereitzustellen.

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

Verwenden Sie die Direktive, um auf eine Einstellung in einer CSHTML-Datei zuzugreifen. `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>
```

Um über den Code auf die in der Datei festgelegten AWS Optionen zuzugreifen, rufen Sie die `GetAWSOptions` Erweiterungsmethode auf, die zu hinzugefügt wurde. `IConfiguration`

Rufen Sie zum Erstellen eines Service-Clients anhand dieser Optionen `CreateServiceClient` auf. Das folgende Beispiel zeigt, wie Sie einen Amazon S3-Serviceclient erstellen. (Achten Sie darauf, Ihrem Projekt das AWSSDK NuGet .S3-Paket hinzuzufügen.)

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

Sie können auch mehrere Service-Clients mit inkompatiblen Einstellungen erstellen, indem Sie mehrere Einträge in der `appsettings.Development.json` Datei verwenden, wie in den folgenden Beispielen gezeigt, in denen die Konfiguration für `service1` die `us-west-2` Region und die Konfiguration für die spezielle Endpunkt-URL *`service2`* enthält.

```
{
  "service1": {
    "Profile": "default",
    "Region": "us-west-2"
  },
  "service2": {
    "Profile": "default",
    "ServiceURL": "URL"
  }
}
```

Anschließend können Sie die Optionen für einen bestimmten Service mithilfe des Eintrags in der JSON-Datei abrufen. `service1`Verwenden Sie beispielsweise Folgendes, um die Einstellungen abzurufen.

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

Zulässige Werte in der Appsettings-Datei

In der `appsettings.Development.json`-Datei können Sie die folgenden Werte für die Anwendungskonfiguration festlegen. Bei den Feldnamen muss die angegebene Groß- und Kleinschreibung verwendet werden. Einzelheiten zu diesen Einstellungen finden Sie in der jeweiligen `AWS.Runtime.ClientConfig` Klasse.

- Region
- Profil
- ProfilesLocation

- SignatureVersion
- RegionEndpoint
- UseHttp
- ServiceURL
- AuthenticationRegion
- AuthenticationServiceName
- MaxErrorRetry
- LogResponse
- BufferSize
- ProgressUpdateInterval
- ResignRetries
- AllowAutoRedirect
- LogMetrics
- DisableLogging
- UseDualstackEndpoint

ASP.NET Core-Abhängigkeitsinjektion

Die AWSSDK.Extensions. NETCore NuGet .Setup-Paket ist auch in ein neues Dependency Injection-System in ASP.NET Core integriert. In der `ConfigureServices` Methode in der `Startup` Klasse Ihrer Anwendung werden die MVC-Dienste hinzugefügt. Wenn die Anwendung Entity Framework verwendet, erfolgt auch die Initialisierung hier.

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

> **Note**
>
> Hintergrundinformationen zur Dependency Injection in .NET Core finden Sie auf der Dokumentationsseite zu .NET Core.

Das `AWSSDK.Extensions.NETCore.Setup` NuGet Paket fügt neue Erweiterungsmethoden hinzu`IServiceCollection`, mit denen Sie AWS Dienste zur Dependency Injection hinzufügen können. Der folgende Code zeigt Ihnen, wie Sie die AWS Optionen hinzufügen, aus denen gelesen wird`IConfiguration`, um Amazon S3 und DynamoDB zur Liste der Dienste hinzuzufügen. (Achten Sie darauf, Ihrem Projekt die DBv2 NuGet Pakete AWSSDK.S3 und AWSSDK.Dynamo hinzuzufügen.)

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

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

Wenn nun Ihre MVC-Controller entweder `IAmazonS3` oder `IAmazonDynamoDB` als Parameter in ihren Konstruktoren verwenden, übergibt das Dependency Injection-System diese Services.

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

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

    ...

}
```

Konfigurieren von anderen Anwendungsparametern

> **Note**
>
> Die Informationen in diesem Thema beziehen sich speziell auf Projekte, die auf .NET Framework basieren. Die `Web.config` Dateien `App.config` und sind in Projekten, die auf .NET Core basieren, standardmäßig nicht vorhanden.

Öffnen, um .NET Framework-Inhalte anzuzeigen

Es gibt eine Reihe von Anwendungsparametern, die Sie konfigurieren können:

- `AWSLogging`
- `AWSLogMetrics`
- `AWSRegion`
- `AWSResponseLogging`
- `AWS.DynamoDBContext.TableNamePrefix`
- `AWS.S3.UseSignatureVersion4`
- `AWSEndpointDefinition`
- AWS Vom Dienst generierte Endpunkte

Diese Parameter können in der `App.config`- oder `Web.config`-Datei der Anwendung konfiguriert werden. Sie können diese zwar auch mit der AWS SDK für .NET API konfigurieren, wir empfehlen jedoch, die Datei der Anwendung zu verwenden. `.config` Nachstehend werden beide Ansätze beschrieben.

Weitere Informationen zur Verwendung des `<aws>` Elements, wie sie später in diesem Thema beschrieben werden, finden Sie unter Referenz für Konfigurationsdateien AWS SDK für .NET.

AWSLogging

Konfiguriert, wie das SDK Ereignisse protokollieren soll, wenn überhaupt. Der Empfohlene Ansatz besteht zum Beispiel darin, das `<logging>`-Element zu verwenden, welches ein untergeordnetes Element des `<aws>`-Elements ist:

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

Alternative Vorgehensweise:

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

Die möglichen Werte sind:

`None`

Deaktivieren Sie die -Ereignisprotokollierung. Dies ist die Standardeinstellung.

`log4net`

Protokollieren mithilfe von log4net.

`SystemDiagnostics`

Protokollieren mithilfe der `System.Diagnostics`-Klasse.

Sie können mehrere Werte für das `logTo`-Attribut festlegen. Diese müssen durch Komma getrennt werden. Im folgenden Beispiel werden die Protokollierung von sowohl `log4net` als auch `System.Diagnostics` in der `.config`-Datei festgelegt:

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

Alternative Vorgehensweise:

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

Kombinieren Sie alternativ mithilfe der AWS SDK für .NET API die Werte der LoggingOptionsAufzählung und legen Sie die AWSConfigs.Logging-Eigenschaft fest:

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

Änderungen an dieser Einstellung werden nur für neue AWS Client-Instances wirksam.

AWSLogMetriken

Gibt an, ob das SDK Leistungsmetriken protokollieren soll oder nicht. Legen Sie zum Einstellen der Konfiguration für die Protokollierung von Metriken in der `.config`-Datei den `logMetrics`-Attributwert im `<logging>`-Element fest, das ein untergeordnetes Element des `<aws>`-Elements ist:

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

Alternativ legen Sie den `AWSLogMetrics`-Schlüssel im Abschnitt `<appSettings>` fest:

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

Um die Protokollierung von Metriken mit der AWS SDK für .NET API einzurichten, legen Sie alternativ den Wert fest AWSConfigs. LogMetricsEigenschaft:

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

Diese Einstellung konfiguriert die standardmäßige `LogMetrics`-Eigenschaft für alle Clients/Konfigurationen. Änderungen an dieser Einstellung werden nur für neue AWS Client-Instances wirksam.

AWSRegion

Konfiguriert die AWS Standardregion für Clients, die nicht explizit eine Region angegeben haben. Zum Einstellen der Region in der `.config`-Datei besteht der empfohlene Ansatz darin, den `region`-Attributwert im `aws`-Element festzulegen:

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

Alternativ legen Sie den AWSRegion-Schlüssel im Abschnitt `<appSettings>` fest:

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

Um die Region mit der AWS SDK für .NET API festzulegen, legen Sie alternativ den AWSConfigs fest. AWSRegionEigenschaft:

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

Weitere Informationen zum Erstellen eines AWS Kunden für eine bestimmte Region finden Sie unter AWS Regionsauswahl. Änderungen an dieser Einstellung werden nur für neue AWS Client-Instances wirksam.

AWSResponseProtokollierung

Konfiguriert, wann das SDK Service-Antworten protokollieren soll. Die möglichen Werte sind:

`Never`

Service-Antworten sollen nie protokolliert werden. Dies ist die Standardeinstellung.

`Always`

Service-Antworten sollen immer protokolliert werden.

`OnError`

Service-Antworten sollen nur im Falle eines Fehlers protokolliert werden.

Zum Einstellen der Konfiguration für die Protokollierung von Service-Antworten in der `.config`-Datei besteht der empfohlene Ansatz darin, den `logResponses`-Attributwert im `<logging>`-Element festzulegen, das ein untergeordnetes Element des `<aws>`-Elements ist:

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

Alternativ können Sie den AWSResponseLogging-Schlüssel im folgenden `<appSettings>` Abschnitt festlegen:

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

Um die Dienstprotokollierung mit der AWS SDK für .NET API einzurichten, legen Sie alternativ den fest AWSConfigs. ResponseLoggingEigenschaft auf einen der Werte der ResponseLoggingOptionAufzählung:

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

Änderungen an dieser Einstellung werden sofort wirksam.

`AWS.DynamoDBContext.TableNamePrefix`

Konfiguriert das standardmäßige `TableNamePrefix`, das `DynamoDBContext` verwendet, sofern es nicht manuell konfiguriert wurde.

Zum Einstellen des Tabellennamen-Präfixes in der `.config`-Datei besteht der empfohlene Ansatz darin, den `tableNamePrefix`-Attributwert im `<dynamoDBContext>`-Element festzulegen, das ein untergeordnetes Element des `<dynamoDB>`-Elements ist, welches selbst ein untergeordnetes Element des `<aws>`-Elements ist:

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

Alternativ legen Sie den `AWS.DynamoDBContext.TableNamePrefix`-Schlüssel im Abschnitt `<appSettings>` fest:

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

Um das Tabellennamenpräfix mit der AWS SDK für .NET API festzulegen, legen Sie alternativ die Eigenschaft AWSConfigs.Dynamo DBContext TableNamePrefix fest:

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

Änderungen an dieser Einstellung werden nur für neu erstellte Instances von `DynamoDBContextConfig` und `DynamoDBContext` wirksam.

`AWS.S3.UseSignatureVersion4`

Konfiguriert, ob der Amazon S3 S3-Client die Signaturversion 4 verwenden soll, um Anfragen zu signieren.

Um die Signaturversion 4-Signatur für Amazon S3 in der `.config` Datei festzulegen, wird empfohlen, das `useSignatureVersion4` Attribut des `<s3>` Elements festzulegen, das ein untergeordnetes `<aws>` Element des Elements ist:

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

Alternativ können Sie den `AWS.S3.UseSignatureVersion4` Schlüssel `true` im `<appSettings>` Abschnitt auf setzen:

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

Um die Signatur der Version 4 mit der AWS SDK für .NET API zu signieren, setzen Sie alternativ die Eigenschaft AWSConfigs.S3 UseSignatureVersion 4 auf`true`:

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

Standardmäßig ist diese Einstellung `false`, Signatur Version 4 kann jedoch in einigen Fällen oder bei einigen Regionen standardmäßig verwendet werden. Wenn die Einstellung `true` ist, wird bei

allen Anfragen Signatur Version 4 verwendet. Änderungen an dieser Einstellung werden nur für neue Amazon S3 S3-Client-Instances wirksam.

AWSEndpointDefinition

Konfiguriert, ob das SDK eine benutzerdefinierte Konfigurationsdatei, die die Regionen und Endpunkte definiert, verwenden soll.

Zum Festlegen der Endpunktdefinitionsdatei in der `.config`-Datei empfehlen wir, den `endpointDefinition`-Attributwert im `<aws>`-Element festzulegen.

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

Alternativ können Sie den AWSEndpointDefinitionsschlüssel im `<appSettings>` Abschnitt festlegen:

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

Um die Endpunktdefinitionsdatei mit der AWS SDK für .NET API festzulegen, legen Sie alternativ den fest AWSConfigs. EndpointDefinitionEigenschaft:

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

Wenn kein Dateiname angegeben wird, wird keine benutzerdefinierte Konfigurationsdatei verwendet. Änderungen an dieser Einstellung werden nur für neue AWS Client-Instances wirksam. Die `endpoints.json` Datei ist verfügbar unterhttps://github.com/aws/aws-sdk-net/blob/main/sdk/src/Core/endpoints.json.

AWS Vom Dienst generierte Endpunkte

Einige AWS Dienste generieren ihre eigenen Endpunkte, anstatt einen regionalen Endpunkt zu nutzen. Clients für diese Services verwenden eine für diesen Service und Ihre Ressourcen spezifische Service-URL. Zwei Beispiele für diese Dienste sind Amazon CloudSearch und AWS IoT. In den folgenden Beispielen wird gezeigt, wie Sie die Endpunkte für diese Services abrufen können.

Beispiel für Amazon CloudSearch Endpoints

Der CloudSearch Amazon-Client wird für den Zugriff auf den CloudSearch Amazon-Konfigurationsservice verwendet. Sie verwenden den CloudSearch Amazon-Konfigurationsservice, um Such-Domains zu erstellen, zu konfigurieren und zu verwalten. Um eine Such-Domain zu erstellen, erstellen Sie ein CreateDomainRequestObjekt und stellen Sie die

`DomainName` Eigenschaft bereit. Erstellen Sie ein AmazonCloudSearchClientObjekt mithilfe des Anforderungsobjekts. Rufen Sie die CreateDomain-Methode auf. Das vom Aufruf zurückgegebene CreateDomainResponseObjekt enthält eine `DomainStatus` Eigenschaft, die `DocService` sowohl den als auch den `SearchService` Endpunkt hat. Erstellen Sie ein AmazonCloudSearchDomainConfigObjekt und verwenden Sie es, um `SearchService` Instanzen der `DocService` AmazonCloudSearchDomainClientKlasse zu initialisieren.

```
// 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 Beispiel für Endpunkte

Um den Endpunkt für zu ermitteln AWS IoT, erstellen Sie ein AmazonIoTClientObjekt und rufen Sie die DescribeEndPointMethode auf. Das zurückgegebene DescribeEndPointResponseObjekt enthält die`EndpointAddress`. Erstellen Sie ein AmazonIotDataConfigObjekt, legen Sie die `ServiceURL` Eigenschaft fest und verwenden Sie das Objekt, um die Klasse zu instanziieren AmazonIotDataClient.

```
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");
}
```

Referenz zu Konfigurationsdateien für AWS SDK für .NET

> **Note**
>
> Die Informationen in diesem Thema beziehen sich speziell auf Projekte, die auf .NET Framework basieren. Die `Web.config` Dateien `App.config` und sind in Projekten, die auf .NET Core basieren, standardmäßig nicht vorhanden.

Öffnen, um .NET Framework-Inhalte anzuzeigen

Sie können ein .NET-Projekt `App.config` oder eine `Web.config` Datei verwenden, um AWS Einstellungen wie AWS Anmeldeinformationen, Protokollierungsoptionen, AWS Service-Endpunkte und AWS Regionen sowie einige Einstellungen für AWS Dienste wie Amazon DynamoDB, Amazon und Amazon S3 EC2 anzugeben. Im Folgenden wird beschrieben, wie die `App.config`- bzw. `Web.config`-Datei ordnungsgemäß zu formatieren ist, um diese Einstellungsarten anzugeben.

> **Note**
>
> Sie können das `<appSettings>` Element in einer `App.config` `Web.config` OR-Datei zwar weiterhin verwenden, um AWS Einstellungen festzulegen, wir empfehlen jedoch, die `<aws>` Elemente `<configSections>` und wie später in diesem Thema beschrieben zu verwenden. Weitere Informationen zu dem `<appSettings>` Element finden Sie in den `<appSettings>` Elementbeispielen unter Konfiguration Ihrer AWS SDK für .NET Anwendung.

> **Note**
>
> Sie können zwar weiterhin die folgenden `AWSConfigs`Klasseneigenschaften in einer Codedatei verwenden, um AWS Einstellungen anzugeben, aber die folgenden Eigenschaften sind veraltet und werden in future Versionen möglicherweise nicht mehr unterstützt:
>
> - `DynamoDBContextTableNamePrefix`

- `EC2UseSignatureVersion4`
- `LoggingOptions`
- `LogMetrics`
- `ResponseLoggingOption`
- `S3UseSignatureVersion4`

Im Allgemeinen empfehlen wir, statt `AWSConfigs` Klasseneigenschaften in einer Codedatei zur Angabe von AWS Einstellungen zu verwenden, sondern die `<aws>` Elemente `<configSections>` und in einer `App.config` `Web.config` OR-Datei zu verwenden, um AWS Einstellungen anzugeben, wie weiter unten in diesem Thema beschrieben. Weitere Informationen zu den oben genannten Eigenschaften finden Sie in den `AWSConfigs` Codebeispielen unter Konfiguration Ihrer AWS SDK für .NET Anwendung.

Themen

- Deklarieren eines AWS Einstellungsabschnitts
- Zulässige Elemente
- Elementreferenz

Deklarieren eines AWS Einstellungsabschnitts

Sie geben AWS Einstellungen in einer `App.config` `Web.config` OR-Datei innerhalb des `<aws>` Elements an. Bevor Sie das `<aws>`-Element verwenden können, müssen Sie ein `<section>`-Element (das ein untergeordnetes Element des `<configSections>`-Elements ist) erstellen und sein `name`-Attribut auf `aws` und sein `type`-Attribut auf `Amazon.AWSSection, AWSSDK.Core` setzen, wie im folgenden Beispiel gezeigt:

```
<?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>
```

Der Visual Studio-Editor bietet keine automatische Codevervollständigung (IntelliSense) für das `<aws>` Element oder seine untergeordneten Elemente.

Unterstützung beim Erstellen einer korrekt formatierten Version des `<aws>`-Elements erhalten Sie durch Aufrufen der `Amazon.AWSConfigs.GenerateConfigTemplate`-Methode. Diese gibt eine kanonische Version des `<aws>`-Elements als eine gut lesbare Zeichenfolge aus, die Sie an Ihre Bedürfnisse anpassen können. In den folgenden Abschnitten werden die Attribute und untergeordneten Elemente des `<aws>`-Elements beschrieben.

Zulässige Elemente

Im Folgenden finden Sie eine Liste der logischen Beziehungen zwischen den zulässigen Elementen in einem AWS Einstellungsbereich. Die aktuelle Version dieser Liste können Sie durch Aufrufen der `Amazon.AWSConfigs.GenerateConfigTemplate`-Methode generieren, die eine kanonische Version des `<aws>`-Elements als Zeichenfolge ausgibt, die Sie an Ihre Bedürfnisse anpassen können.

```
<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>
```

Elementreferenz

Im Folgenden finden Sie eine Liste der Elemente, die in einem AWS Einstellungsbereich zulässig sind. Für jedes Element werden die zulässigen Attribute und die unter- und übergeordneten Elemente aufgelistet.

Themen

- alias
- aws
- dynamoDB
- Dynamo DBContext
- ec2
- Protokollierung
- map
- property
- Proxy
- S3

alias

Das <alias>-Element repräsentiert ein einzelnes Element in einer Sammlung von einer oder mehreren From-Table-To-Table-Zuweisungen, das eine andere als eine für einen Typ konfigurierte Tabelle angibt. Dieses Element entspricht einer Instance der Amazon.Util.TableAlias-Klasse aus der Amazon.AWSConfigs.DynamoDBConfig.Context.TableAliases-Eigenschaft im AWS SDK für .NET. Die Neuzuweisung erfolgt vor der Anwendung eines Tabellennamen-Präfixes.

Dieses Element kann folgende Attribute enthalten:

fromTable

From-Table-Teil der From-Table-To-Table-Zuweisung. Dieses Attribut entspricht der Amazon.Util.TableAlias.FromTable-Eigenschaft im AWS SDK für .NET.

toTable

To-Table-Teil der From-Table-To-Table-Zuweisung. Dieses Attribut entspricht der Amazon.Util.TableAlias.ToTable-Eigenschaft im AWS SDK für .NET.

Das übergeordnete Element des <alias>-Elements ist das <tableAliases>-Element.

Das <alias>-Element enthält keine untergeordneten Elemente.

Nachstehend finden Sie ein Beispiel des <alias>-Elements in Verwendung:

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

aws

Das <aws> Element stellt das oberste Element in einem AWS Einstellungsbereich dar. Dieses Element kann folgende Attribute enthalten:

endpointDefinition

Der absolute Pfad zu einer benutzerdefinierten Konfigurationsdatei, die die zu verwendenden AWS Regionen und Endpunkte definiert. Dieses Attribut entspricht der Amazon.AWSConfigs.EndpointDefinition-Eigenschaft im AWS SDK für .NET.

`profileName`

Der Profilname für gespeicherte AWS Anmeldeinformationen, die für Serviceanfragen verwendet werden. Dieses Attribut entspricht der `Amazon.AWSConfigs.AWSProfileName`-Eigenschaft im AWS SDK für .NET.

`profilesLocation`

Der absolute Pfad zum Speicherort der Anmeldeinformationsdatei, die mit anderen geteilt wurde AWS SDKs. Standardmäßig wird die Anmeldeinformationsdatei im `.aws`-Verzeichnis im Basisordner des aktuellen Benutzers gespeichert. Dieses Attribut entspricht der `Amazon.AWSConfigs.AWSProfilesLocation`-Eigenschaft im AWS SDK für .NET.

`region`

Die AWS Standardregions-ID für Clients, die nicht explizit eine Region angegeben haben. Dieses Attribut entspricht der `Amazon.AWSConfigs.AWSRegion`-Eigenschaft im AWS SDK für .NET.

Das `<aws>`-Element hat kein übergeordnetes Element.

Das `<aws>`-Element kann die folgenden untergeordneten Elemente enthalten:

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

Nachstehend finden Sie ein Beispiel des `<aws>`-Elements in Verwendung:

```
<aws
  endpointDefinition="C:\Configs\endpoints.xml"
  region="us-west-2"
  profileName="development"
  profilesLocation="C:\Configs">
  <!-- ... -->
</aws>
```

dynamoDB

Das <dynamoDB>-Element repräsentiert eine Sammlung von Einstellungen für Amazon DynamoDB. Dieses Element kann das Attribut conversionSchema enthalten, welches die für die Konvertierung zwischen .NET- und DynamoDB-Objekten zu verwendende Version repräsentiert. Zulässige Werte sind V1 und V2. Dieses Attribut entspricht der `Amazon.DynamoDBv2.DynamoDBEntryConversion`-Klasse im AWS SDK für .NET. Weitere Informationen finden Sie unter DynamoDB-Serien – Konvertierungsschemas.

Das übergeordnete Element des <dynamoDB>-Elements ist das <aws>-Element.

Das <dynamoDB>-Element kann das untergeordnete <dynamoDBContext>-Element enthalten.

Nachstehend finden Sie ein Beispiel des <dynamoDB>-Elements in Verwendung:

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

Dynamo DBContext

Das <dynamoDBContext>-Element repräsentiert eine Sammlung von kontextspezifischen Einstellungen für Amazon DynamoDB. Dieses Element kann das tableNamePrefixAttribut enthalten, das das standardmäßige Tabellennamenpräfix darstellt, das der DynamoDB-Kontext verwendet, wenn er nicht manuell konfiguriert wird. Dieses Attribut entspricht der `Amazon.Util.DynamoDBContextConfig.TableNamePrefix`-Eigenschaft aus der `Amazon.AWSConfigs.DynamoDBConfig.Context.TableNamePrefix`-Eigenschaft im AWS SDK für .NET. Weitere Informationen finden Sie unter Enhancements to the DynamoDB SDK.

Das übergeordnete Element des <dynamoDBContext>-Elements ist das <dynamoDB>-Element.

Das <dynamoDBContext>-Element kann die folgenden untergeordneten Elemente enthalten:

- <alias> (eine oder mehrere Instances)
- <map> (eine oder mehrere Instances)

Nachstehend finden Sie ein Beispiel des <dynamoDBContext>-Elements in Verwendung:

```
<dynamoDBContext
```

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

ec2

Das <ec2> Element stellt eine Sammlung von EC2 Amazon-Einstellungen dar. Dieses Element kann das useSignatureVersion4-Attribut enthalten, das angibt, ob die Signatur der Version 4 für alle Anfragen verwendet wird (true) oder ob die Signatur der Version 4 nicht für alle Anfragen verwendet wird (false, Standardeinstellung). Dieses Attribut entspricht der `Amazon.Util.EC2Config.UseSignatureVersion4`-Eigenschaft aus der `Amazon.AWSConfigs.EC2Config.UseSignatureVersion4`-Eigenschaft im AWS SDK für .NET.

Das übergeordnete Element des <ec2>-Elements ist das-Element.

Das <ec2>-Element enthält keine untergeordneten Elemente.

Nachstehend finden Sie ein Beispiel des <ec2>-Elements in Verwendung:

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

Protokollierung

Das <logging>-Element repräsentiert eine Sammlung von Einstellungen für die Protokollierung von Antworten und Leistungsmetriken. Dieses Element kann folgende Attribute enthalten:

`logMetrics`

Legt fest, ob die Protokollierung der Leistungsmetriken für alle Clients und Konfigurationen (true) erfolgt, oder nicht (false). Dieses Attribut entspricht der `Amazon.Util.LoggingConfig.LogMetrics`-Eigenschaft aus der `Amazon.AWSConfigs.LoggingConfig.LogMetrics`-Eigenschaft im AWS SDK für .NET.

`logMetricsCustomFormatter`

Der Datentyp und der Assembly-Name eines benutzerdefinierten Formatierers für die Protokollierung von Metriken. Dieses Attribut entspricht der `Amazon.Util.LoggingConfig.LogMetricsCustomFormatter`-Eigenschaft aus der `Amazon.AWSConfigs.LoggingConfig.LogMetricsCustomFormatter`-Eigenschaft im AWS SDK für .NET.

`logMetricsFormat`

Das Format, in dem die protokollierten Metriken dargestellt sind (entspricht der `Amazon.Util.LoggingConfig.LogMetricsFormat`-Eigenschaft aus der `Amazon.AWSConfigs.LoggingConfig.LogMetricsFormat`-Eigenschaft im AWS SDK für .NET).

Gültige Werte sind:

`JSON`

Verwenden Sie das JSON-Format.

`Standard`

Verwenden Sie das Standardformat.

`logResponses`

Legt fest, wann Service-Antworten protokolliert werden sollen (entspricht der `Amazon.Util.LoggingConfig.LogResponses`-Eigenschaft aus der `Amazon.AWSConfigs.LoggingConfig.LogResponses`-Eigenschaft im AWS SDK für .NET).

Gültige Werte sind:

`Always`

Service-Antworten sollen immer protokolliert werden.

`Never`

Service-Antworten sollen nie protokolliert werden.

`OnError`

Service-Antworten sollen nur im Falle von Fehlern protokolliert werden.

`logTo`

Wo soll ich mich anmelden (ordnet der `LogTo` Immobilie von der `Amazon.AWSConfigs.LoggingConfig.LogTo` Unterkunft aus zu AWS SDK für .NET).

Die zulässigen Werte sind:

`Log4Net`

Das Protokoll wird in log4net geschrieben.

`None`

Die Protokollierung ist deaktiviert.

`SystemDiagnostics`

Das Protokoll wird in System.Diagnostics geschrieben.

Das übergeordnete Element des `<logging>`-Elements ist das `<aws>`-Element.

Das `<logging>`-Element enthält keine untergeordneten Elemente.

Nachstehend finden Sie ein Beispiel des `<logging>`-Elements in Verwendung:

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

map

Das `<map>` Element stellt ein einzelnes Element in einer Sammlung von type-to-table Zuordnungen von.NET-Typen zu DynamoDB-Tabellen dar (wird einer Instanz der `TypeMapping` Klasse aus der `Amazon.AWSConfigs.DynamoDBConfig.Context.TypeMappings` Eigenschaft in zugeordnet). AWS SDK für .NET Weitere Informationen finden Sie unter Enhancements to the DynamoDB SDK.

Dieses Element kann folgende Attribute enthalten:

`targetTable`

Die DynamoDB-Tabelle, auf die die Zuweisung angewendet wird. Dieses Attribut entspricht der `Amazon.Util.TypeMapping.TargetTable`-Eigenschaft im AWS SDK für .NET.

`type`

Der Typ und Assembly-Name, auf den die Zuweisung angewendet wird. Dieses Attribut entspricht der `Amazon.Util.TypeMapping.Type`-Eigenschaft im AWS SDK für .NET.

Das übergeordnete Element des `<map>`-Elements ist das `<dynamoDBContext>`-Element.

Das <map>-Element kann eine oder mehrere Instances des untergeordneten <property>-Elements enthalten.

Nachstehend finden Sie ein Beispiel des <map>-Elements in Verwendung:

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

property

Das <property>-Element repräsentiert eine DynamoDB-Eigenschaft. (Dieses Element ist einer Instanz von Amazon.Util zugeordnet. PropertyConfig Klasse aus der AddProperty Methode in AWS SDK für .NET) Weitere Informationen finden Sie unter Erweiterungen des DynamoDB-SDK und der DynamoDB-Attribute.

Dieses Element kann folgende Attribute enthalten:

attribute

Der Name eines Attributs für die Eigenschaft, z. B. der Name eines Bereichsschlüssels. Dieses Attribut entspricht der Amazon.Util.PropertyConfig.Attribute-Eigenschaft im AWS SDK für .NET.

converter

Der Typ des Konverters, der für diese Eigenschaft verwendet werden soll. Dieses Attribut entspricht der Amazon.Util.PropertyConfig.Converter-Eigenschaft im AWS SDK für .NET.

ignore

Legt fest, ob die zugeordnete Eigenschaft ignoriert werden soll (true) oder nicht (false). Dieses Attribut entspricht der Amazon.Util.PropertyConfig.Ignore-Eigenschaft im AWS SDK für .NET.

name

Der Name der Eigenschaft. Dieses Attribut entspricht der Amazon.Util.PropertyConfig.Name-Eigenschaft im AWS SDK für .NET.

`version`

Legt fest, ob diese Eigenschaft die Versionsnummer des Elements speichern soll (true) oder nicht (false). Dieses Attribut entspricht der `Amazon.Util.PropertyConfig.Version`-Eigenschaft im AWS SDK für .NET.

Das übergeordnete Element des `<property>`-Elements ist das `<map>`-Element.

Das `<property>`-Element enthält keine untergeordneten Elemente.

Nachstehend finden Sie ein Beispiel des `<property>`-Elements in Verwendung:

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

Proxy

Das `<proxy>`-Element repräsentiert Einstellungen für die Konfiguration eines Proxys für das AWS SDK für .NET . Dieses Element kann folgende Attribute enthalten:

Host

Der Hostname oder die IP-Adresse des Proxy-Servers. Dieses Attribut entspricht der `Amazon.Util.ProxyConfig.Host`-Eigenschaft aus der `Amazon.AWSConfigs.ProxyConfig.Host`-Eigenschaft im AWS SDK für .NET.

password

Das Passwort für die Authentifizierung mit dem Proxy-Server. Dieses Attribut entspricht der `Amazon.Util.ProxyConfig.Password`-Eigenschaft aus der `Amazon.AWSConfigs.ProxyConfig.Password`-Eigenschaft im AWS SDK für .NET.

port

Die Portnummer des Proxys. Dieses Attribut entspricht der `Amazon.Util.ProxyConfig.Port`-Eigenschaft aus der `Amazon.AWSConfigs.ProxyConfig.Port`-Eigenschaft im AWS SDK für .NET.

username

Der Benutzername für die Authentifizierung mit dem Proxy-Server. Dieses Attribut entspricht der `Amazon.Util.ProxyConfig.Username`-Eigenschaft aus der `Amazon.AWSConfigs.ProxyConfig.Username`-Eigenschaft im AWS SDK für .NET.

Das übergeordnete Element des `<proxy>`-Elements ist das `<aws>`-Element.

Das `<proxy>`-Element enthält keine untergeordneten Elemente.

Nachstehend finden Sie ein Beispiel des `<proxy>`-Elements in Verwendung:

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

S3

Das `<s3>`-Element repräsentiert eine Sammlung von Einstellungen für Amazon S3. Dieses Element kann das useSignatureVersion4-Attribut enthalten, das angibt, ob die Signatur der Version 4 für alle Anfragen verwendet wird (true) oder ob die Signatur der Version 4 nicht für alle Anfragen verwendet wird (false, Standardeinstellung). Dieses Attribut entspricht der `Amazon.AWSConfigs.S3Config.UseSignatureVersion4`-Eigenschaft im AWS SDK für .NET.

Das übergeordnete Element des `<s3>`-Elements ist das `<aws>`-Element.

Das `<s3>`-Element enthält keine untergeordneten Elemente.

Nachstehend finden Sie ein Beispiel des `<s3>`-Elements in Verwendung:

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

Verwenden von Legacy-Anmeldeinformationen

Die Themen in diesem Abschnitt enthalten Informationen zur Verwendung von lang- oder kurzfristigen Anmeldeinformationen ohne Verwendung von AWS IAM Identity Center.

> **Warning**
>
> Um Sicherheitsrisiken zu vermeiden, sollten Sie IAM-Benutzer nicht zur Authentifizierung verwenden, wenn Sie speziell entwickelte Software entwickeln oder mit echten Daten arbeiten. Verwenden Sie stattdessen den Verbund mit einem Identitätsanbieter wie AWS IAM Identity Center.

> **Note**
>
> Die Informationen in diesem Thema beziehen sich auf Situationen, in denen Sie kurz- oder langfristige Anmeldeinformationen manuell abrufen und verwalten müssen. Weitere Informationen zu kurz- und langfristigen Anmeldeinformationen finden Sie unter Andere Authentifizierungsmethoden im AWS SDKs Referenzhandbuch zu Tools.
> Bewährte Sicherheitspraktiken finden Sie AWS IAM Identity Center wie unter beschriebenAuthentifizierung mit AWS.

Wichtige Warnhinweise und Richtlinien für Anmeldeinformationen

Warnhinweise für Anmeldeinformationen

- Verwenden Sie NICHT die Root-Anmeldeinformationen Ihres Kontos, um auf AWS Ressourcen zuzugreifen. Diese Anmeldeinformationen bieten uneingeschränkten Zugriff auf Konten und können nur schwer widerrufen werden.
- Geben Sie KEINE wörtlichen Zugriffsschlüssel oder Anmeldeinformationen in Ihre Anwendungsdateien ein. Wenn Sie dies tun, riskieren Sie damit, dass Ihre Kontodaten versehentlich offengelegt werden, falls Sie z. B. das Projekt in ein öffentliches Repository hochladen.
- Fügen Sie KEINE Dateien in Ihrem Projektbereich hinzu, die Anmeldeinformationen enthalten.
- Beachten Sie, dass alle Anmeldeinformationen, die in der gemeinsam genutzten AWS `credentials` Datei gespeichert sind, im Klartext gespeichert werden.

Zusätzliche Hinweise zur sicheren Verwaltung von Anmeldeinformationen

Eine allgemeine Erläuterung der sicheren Verwaltung von AWS Anmeldeinformationen finden Sie unter AWS Sicherheitsanmeldedaten im Allgemeine AWS-ReferenzIAM-Benutzerhandbuch und

Bewährte Sicherheitsmethoden und Anwendungsfälle. Berücksichtigen Sie zusätzlich zu diesen Informationen Folgendes:

- Erstellen Sie zusätzliche Benutzer, z. B. Benutzer in IAM Identity Center, und verwenden Sie deren Anmeldeinformationen anstelle Ihrer AWS -Root-Benutzeranmeldeinformationen. Anmeldeinformationen für andere Benutzer können bei Bedarf widerrufen werden oder sind temporärer Natur. Darüber hinaus können Sie auf jeden Benutzer eine Richtlinie anwenden, die nur den Zugriff auf bestimmte Ressourcen und Aktionen vorsieht und so eine Einstellung der Rechte mit den geringsten Berechtigungen einräumt.

- Verwenden Sie IAM-Rollen für Aufgaben in Verbindung mit Aufgaben von Amazon Elastic Container Service (Amazon ECS).

- Verwenden Sie IAM-Rollen für Anwendungen, die auf EC2 Amazon-Instances ausgeführt werden.

- Verwenden Sie temporäre Anmeldeinformationen oder Umgebungsvariablen für Anwendungen, die Benutzern außerhalb Ihrer Organisation zur Verfügung stehen.

Themen

- Die Datei mit den gemeinsam genutzten AWS Anmeldeinformationen verwenden
- Den SDK Store verwenden (nur Windows)

Die Datei mit den gemeinsam genutzten AWS Anmeldeinformationen verwenden

(Lesen Sie unbedingt die wichtigen Warnungen und Anleitungen zu den Zugangsdaten.)

Eine Möglichkeit, Anmeldeinformationen für Ihre Anwendungen bereitzustellen, besteht darin, Profile in der Datei mit gemeinsamen AWS Anmeldeinformationen zu erstellen und die Anmeldeinformationen dann in diesen Profilen zu speichern. Diese Datei kann von der anderen verwendet werden AWS SDKs. Sie kann auch von den AWS CLIAWS Tools for Windows PowerShell, und den AWS Toolkits für Visual Studio und VS Code verwendet werden. JetBrains

> **Warning**
>
> Um Sicherheitsrisiken zu vermeiden, sollten Sie IAM-Benutzer nicht zur Authentifizierung verwenden, wenn Sie speziell entwickelte Software entwickeln oder mit echten Daten arbeiten. Verwenden Sie stattdessen den Verbund mit einem Identitätsanbieter wie AWS IAM Identity Center.

> **Note**
>
> Die Informationen in diesem Thema beziehen sich auf Situationen, in denen Sie kurz- oder langfristige Anmeldeinformationen manuell abrufen und verwalten müssen. Weitere Informationen zu kurz- und langfristigen Anmeldeinformationen finden Sie unter Andere Authentifizierungsmethoden im Referenzhandbuch AWS SDKs und im Tools-Referenzhandbuch.
> Bewährte Sicherheitspraktiken finden Sie AWS IAM Identity Center wie unter beschriebenAuthentifizierung mit AWS.

Allgemeine Informationen

Standardmäßig befindet sich die Datei mit den gemeinsamen AWS Anmeldeinformationen in dem `.aws` Verzeichnis in Ihrem Home-Verzeichnis und hat einen Namen, `credentials` d. h. `~/.aws/credentials` (Linux oder macOS) oder `%USERPROFILE%\.aws\credentials` (Windows). Informationen zu alternativen Speicherorten finden Sie unter Speicherort der gemeinsam genutzten Dateien im AWS SDKs Tools-Referenzhandbuch. Lesen Sie auch Zugreifen auf Anmeldeinformationen und Profile in einer Anwendung.

Die Datei mit den gemeinsamen AWS Anmeldeinformationen ist eine Klartextdatei und folgt einem bestimmten Format. Informationen zum Format von AWS Anmeldeinformationsdateien finden Sie unter Format der Anmeldeinformationsdatei im Referenzhandbuch AWS SDKs und im Tools-Referenzhandbuch.

Sie können die Profile in der Datei mit den gemeinsamen AWS Anmeldeinformationen auf verschiedene Arten verwalten.

- Verwenden Sie einen beliebigen Texteditor, um die Datei mit den gemeinsamen AWS Anmeldeinformationen zu erstellen und zu aktualisieren.

- Verwenden Sie die Amazon.Runtime. CredentialManagementNamespace der AWS SDK für .NET API, wie später in diesem Thema gezeigt wird.
- Verwenden Sie Befehle und Verfahren für die AWS -Tools für PowerShellund die AWS Toolkits für Visual Studio und VS Code. JetBrains
- Verwenden Sie AWS CLIBefehle, z. B. `aws configure set aws_access_key_id` und`aws configure set aws_secret_access_key`.

Beispiele für Profilverwaltung

Die folgenden Abschnitte zeigen Beispiele für Profile in der Datei mit gemeinsamen AWS Anmeldeinformationen. Einige der Beispiele zeigen das Ergebnis, das mit jeder der zuvor beschriebenen Methoden zur Verwaltung von Anmeldeinformationen erzielt werden kann. Andere Beispiele zeigen, wie eine bestimmte Methode verwendet wird.

Das Standardprofil

Die Datei mit gemeinsamen AWS Anmeldeinformationen hat fast immer ein Profil mit dem Namen default. Hier AWS SDK für .NET sucht der nach Anmeldeinformationen, wenn keine anderen Profile definiert sind.

Das `[default]` Profil sieht normalerweise in etwa wie folgt aus.

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

Erstellen Sie programmgesteuert ein Profil

In diesem Beispiel wird gezeigt, wie Sie ein Profil erstellen und es programmgesteuert in der Datei mit gemeinsamen AWS Anmeldeinformationen speichern. Es verwendet die folgenden Klassen der Amazon.Runtime. CredentialManagementNamespace: CredentialProfileOptions, und CredentialProfile. 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**
>
> Code wie dieser sollte im Allgemeinen nicht in Ihrer Anwendung enthalten sein. Wenn Sie ihn in Ihre Anwendung aufnehmen, treffen Sie geeignete Vorkehrungen, um sicherzustellen, dass Klartext-Schlüssel unmöglich im Code, über das Netzwerk oder sogar im Computerspeicher zu sehen sind.

Im Folgenden finden Sie das Profil, das anhand dieses Beispiels erstellt wurde.

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

Aktualisieren Sie ein vorhandenes Profil programmgesteuert

Dieses Beispiel zeigt Ihnen, wie Sie das zuvor erstellte Profil programmgesteuert aktualisieren können. Es verwendet die folgenden Klassen der Amazon.Runtime.CredentialManagementNamespace: und. CredentialProfileSharedCredentialsFile Es verwendet auch die RegionEndpointKlasse des Amazon-Namespace.

```
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);
    }
}
```

Das Folgende ist das aktualisierte Profil.

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

> **Note**
>
> Sie können die AWS Region auch an anderen Orten und mit anderen Methoden festlegen. Weitere Informationen finden Sie unter Festlegung der AWS Region für AWS SDK für .NET.

Den SDK Store verwenden (nur Windows)

(Lesen Sie unbedingt die wichtigen Warnungen und Richtlinien.)

Unter Windows ist der SDK Store ein weiterer Ort, an dem Sie Profile erstellen und verschlüsselte Anmeldeinformationen für Ihre AWS SDK für .NET Anwendung speichern können. Er befindet sich in`%USERPROFILE%\AppData\Local\AWSToolkit\RegisteredAccounts.json`. Sie können den SDK Store während der Entwicklung als Alternative zur Datei mit gemeinsam genutzten AWS Anmeldeinformationen verwenden.

> **Warning**
>
> Um Sicherheitsrisiken zu vermeiden, sollten Sie IAM-Benutzer nicht zur Authentifizierung verwenden, wenn Sie speziell entwickelte Software entwickeln oder mit echten Daten

arbeiten. Verwenden Sie stattdessen den Verbund mit einem Identitätsanbieter wie AWS IAM Identity Center.

> **Note**
>
> Die Informationen in diesem Thema beziehen sich auf Situationen, in denen Sie kurz- oder langfristige Anmeldeinformationen manuell abrufen und verwalten müssen. Weitere Informationen zu kurz- und langfristigen Anmeldeinformationen finden Sie unter Andere Authentifizierungsmethoden im AWS SDKs Referenzhandbuch zu Tools.
> Bewährte Sicherheitspraktiken finden Sie AWS IAM Identity Center wie unter beschriebenAuthentifizierung mit AWS.

Allgemeine Informationen

Der SDK Store bietet die folgenden Vorteile:

- Die Anmeldeinformationen im SDK-Store sind verschlüsselt, und der SDK-Store befindet sich im Home-Verzeichnis des Benutzers. Dadurch wird das Risiko einer versehentlichen Offenlegung Ihrer Anmeldeinformationen begrenzt.
- Der SDK-Store stellt auch Anmeldeinformationen für den AWS Tools for Windows PowerShellund den AWS Toolkit for Visual Studiobereit.

SDK Store-Profile sind spezifisch für einen bestimmten Benutzer auf einem bestimmten Host. Sie können sie nicht auf andere Hosts oder andere Benutzer kopieren. Das bedeutet, dass Sie SDK Store-Profile, die sich auf Ihrem Entwicklungscomputer befinden, nicht für andere Hosts oder Entwicklercomputer wiederverwenden können. Das bedeutet auch, dass Sie SDK Store-Profile nicht in Produktionsanwendungen verwenden können.

Sie können die Profile im SDK Store auf folgende Weise verwalten:

- Verwenden Sie die grafische Benutzeroberfläche (GUI) in der AWS Toolkit for Visual Studio.
- Verwenden Sie die Amazon.Runtime. CredentialManagementNamespace der AWS SDK für .NET API, wie später in diesem Thema gezeigt wird.
- Verwenden Sie Befehle aus dem AWS Tools for Windows PowerShell; zum Beispiel `Set-AWSCredential` und`Remove-AWSCredentialProfile`.

Beispiele für Profilverwaltung

Die folgenden Beispiele zeigen, wie Sie ein Profil im SDK Store programmgesteuert erstellen und aktualisieren.

Programmgesteuert ein Profil erstellen

Dieses Beispiel zeigt Ihnen, wie Sie ein Profil erstellen und es programmgesteuert im SDK Store speichern. Es verwendet die folgenden Klassen der Amazon.Runtime. CredentialManagementNamespace: CredentialProfileOptionsCredentialProfile, und Net File. SDKCredentials

```
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**
>
> Code wie dieser sollte im Allgemeinen nicht in Ihrer Anwendung enthalten sein. Wenn er in Ihrer Anwendung enthalten ist, treffen Sie geeignete Vorkehrungen, um sicherzustellen, dass Klartext-Schlüssel unmöglich im Code, über das Netzwerk oder sogar im Computerspeicher zu sehen sind.

Das folgende Profil wurde anhand dieses Beispiels erstellt.

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

Aktualisieren Sie ein vorhandenes Profil programmgesteuert

Dieses Beispiel zeigt Ihnen, wie Sie das zuvor erstellte Profil programmgesteuert aktualisieren können. Es verwendet die folgenden Klassen der Amazon.Runtime. CredentialManagementNamespace: CredentialProfileund Net File. SDKCredentials Es verwendet auch die RegionEndpointKlasse des Amazon-Namespace.

```
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);
    }
}
```

Das Folgende ist das aktualisierte Profil.

```
"[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**
>
> Sie können die AWS Region auch an anderen Orten und mit anderen Methoden festlegen. Weitere Informationen finden Sie unter Festlegung der AWS Region für AWS SDK für .NET.

Mit dem AWS SDK für .NET

Dieser Abschnitt enthält Informationen zu Funktionen von AWS SDK für .NET , die Sie bei der Erstellung Ihrer Anwendungen möglicherweise berücksichtigen müssen.

Bevor Sie beginnen, stellen Sie sicher, dass Sie Ihre Umgebung eingerichtet und Ihr Projekt konfiguriert haben.

Informationen zur Entwicklung von Software für bestimmte AWS Dienste sowie Codebeispiele finden Sie unter AWS Dienste anrufen. Weitere Codebeispiele finden Sie unterSDK für .NET (v4) Codebeispiele.

Themen

- Asynchrone Programmierung mit dem AWS SDK für .NET
- Verwenden von paginierten Ergebnissen in der AWS SDK für .NET
- Support für HTTP 2 in der AWS SDK für .NET
- Zusätzliche Tools

Asynchrone Programmierung mit dem AWS SDK für .NET

Das AWS SDK für .NET verwendet das Task-Based Asynchronous Pattern (TAP) für seine asynchrone Implementierung. Weitere Informationen zum TAP finden Sie unter Task-based Asynchronous Pattern (TAP) auf docs.microsoft.com.

Dieses Thema gibt Ihnen einen Überblick darüber, wie Sie TAP bei Ihren Anrufen an Serviceclients verwenden können. AWS

Die asynchronen Methoden in der AWS SDK für .NET API sind Operationen, die auf der `Task` Klasse oder der `Task<TResult>` Klasse basieren. Informationen zu diesen Klassen finden Sie auf docs.microsoft.com: Taskclass, Task< > class. TResult

Wenn diese API-Methoden in Ihrem Code aufgerufen werden, müssen sie innerhalb einer Funktion aufgerufen werden, die mit dem `async` Schlüsselwort deklariert ist, wie im folgenden Beispiel gezeigt.

```
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;
}
```

Wie im vorherigen Codeausschnitt gezeigt, ist der bevorzugte Bereich für die `async` Deklaration die Funktion. `Main` Durch die Festlegung dieses `async` Bereichs wird sichergestellt, dass alle Aufrufe an AWS Service-Clients asynchron sein müssen. Wenn Sie aus irgendeinem Grund nicht als asynchron deklarieren `Main` können, können Sie das `async` Schlüsselwort für andere Funktionen als verwenden `Main` und dann die API-Methoden von dort aus aufrufen, wie im folgenden Beispiel gezeigt.

```
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;
}
```

Beachten Sie die spezielle `Task<>` Syntax, die erforderlich ist`Main`, wenn Sie dieses Muster verwenden. Außerdem müssen Sie das **`Result`**Mitglied der Antwort verwenden, um die Daten abzurufen.

Vollständige Beispiele für asynchrone Aufrufe von AWS Service-Clients finden Sie in den Erstellen einer einfachen Anwendung Abschnitten (Einfache plattformübergreifende AppundEinfache Windows-basierte App) und inGeführte Codebeispiele.

Verwenden von paginierten Ergebnissen in der AWS SDK für .NET

Einige AWS Dienste sammeln und speichern eine große Datenmenge, die Sie mithilfe der API-Aufrufe von abrufen können. AWS SDK für .NET Wenn die Datenmenge, die Sie abrufen möchten, für einen einzelnen API-Aufruf zu groß wird, können Sie die Ergebnisse mithilfe der Paginierung in überschaubarere Teile aufteilen.

Damit Sie die Paginierung durchführen können, stellen die Anforderungs- und Antwortobjekte für viele Service-Clients im SDK ein Fortsetzungstoken (normalerweise benannt) bereit. `NextToken` Einige dieser Service-Clients bieten auch Paginatoren.

Mit Paginatoren können Sie den Mehraufwand vermeiden, der mit dem Fortsetzungstoken verbunden ist, der Schleifen, Statusvariablen, mehrere API-Aufrufe usw. beinhalten kann. Wenn Sie einen Paginator verwenden, können Sie Daten aus einem AWS Dienst über eine einzige Codezeile, die Deklaration einer Schleife, abrufen. `foreach` Wenn mehrere API-Aufrufe erforderlich sind, um die Daten abzurufen, erledigt der Paginator dies für Sie.

Wo finde ich Paginatoren?

Nicht alle Dienste bieten Paginatoren. Eine Möglichkeit, festzustellen, ob ein Dienst einen Paginator für eine bestimmte API bereitstellt, besteht darin, sich die Definition einer Service-Client-Klasse in der API-Referenz anzusehen.AWS SDK für .NET

Wenn Sie beispielsweise die Definition für die AmazonCloudWatchLogsClientKlasse untersuchen, sehen Sie eine `Paginators` Eigenschaft. Dies ist die Eigenschaft, die einen Paginator für Amazon CloudWatch Logs bereitstellt.

Was geben mir Paginatoren?

Paginatoren enthalten Eigenschaften, die es Ihnen ermöglichen, vollständige Antworten zu sehen. Sie enthalten in der Regel auch eine oder mehrere Eigenschaften, die es Ihnen ermöglichen, auf die interessantesten Teile der Antworten zuzugreifen, die wir die wichtigsten Ergebnisse nennen werden.

In den zuvor `AmazonCloudWatchLogsClient` genannten Fällen enthält das `Paginator` Objekt beispielsweise eine `Responses` Eigenschaft mit dem vollständigen

DescribeLogGroupsResponseObjekt aus dem API-Aufruf. Diese `Responses` Eigenschaft enthält unter anderem eine Sammlung der Protokollgruppen.

Das Paginator-Objekt enthält auch ein wichtiges Ergebnis mit dem Namen. `LogGroups` Diese Eigenschaft enthält nur den Teil der Antwort mit den Protokollgruppen. Dieses wichtige Ergebnis ermöglicht es Ihnen, Ihren Code unter vielen Umständen zu reduzieren und zu vereinfachen.

Synchrone und asynchrone Paginierung

Paginatoren bieten sowohl synchrone als auch asynchrone Mechanismen für die Paginierung. Synchrone Paginierung ist in Projekten von.NET Framework 4.7.2 (oder höher) verfügbar. Asynchrone Paginierung ist in .NET Core-Projekten (.NET Core 3.1, .NET 5 usw.) verfügbar.

Da asynchrone Operationen und .NET Core empfohlen werden, zeigt Ihnen das folgende Beispiel die asynchrone Paginierung. Informationen dazu, wie Sie dieselben Aufgaben mit synchroner Paginierung und .NET Framework 4.7.2 (oder höher) ausführen können, finden Sie nach dem Beispiel unter. Zusätzliche Überlegungen zu Paginatoren

Beispiel

Das folgende Beispiel zeigt Ihnen, wie Sie mit dem AWS SDK für .NET eine Liste von Protokollgruppen anzeigen können. Im Gegensatz dazu zeigt das Beispiel, wie Sie dies sowohl mit als auch ohne Paginatoren tun können. Bevor Sie sich den vollständigen Code ansehen, der später gezeigt wird, sollten Sie sich die folgenden Ausschnitte ansehen.

CloudWatch Loggruppen ohne Paginatoren abrufen

```
// 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));
```

Abrufen von CloudWatch Protokollgruppen mithilfe von Paginatoren

```
    // 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);
    }
```

Die Ergebnisse dieser beiden Snippets sind exakt identisch, sodass der Vorteil der Verwendung von Paginatoren deutlich zu erkennen ist.

> **Note**
>
> Bevor Sie versuchen, den vollständigen Code zu erstellen und auszuführen, stellen Sie sicher, dass Sie Ihre Umgebung und Ihr Projekt eingerichtet haben.
> Möglicherweise benötigen Sie auch die Microsoft.Bcl. AsyncInterfaces NuGet Paket, weil asynchrone Paginatoren die Schnittstelle verwenden. `IAsyncEnumerable`

Vollständiger Code

Dieser Abschnitt enthält relevante Referenzen und den vollständigen Code für dieses Beispiel.

SDK-Referenzen

NuGet Pakete:

- AWSSDK.CloudWatch

Elemente der Programmierung:

- Namespace Amazon. CloudWatch

  Klasse AmazonCloudWatchLogsClient
- Namespace Amazon. CloudWatchLogs.Modell

  Klasse DescribeLogGroupsRequest

  Klasse DescribeLogGroupsResponse

Klasse LogGroup

Vollständiger Code

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

Zusätzliche Überlegungen zu Paginatoren

- Paginatoren können nicht mehr als einmal verwendet werden

  Wenn Sie die Ergebnisse eines bestimmten AWS Paginators an mehreren Stellen in Ihrem Code benötigen, dürfen Sie ein Paginator-Objekt nicht mehr als einmal verwenden. Erstellen Sie stattdessen jedes Mal, wenn Sie ihn benötigen, einen neuen Paginator. Dieses Konzept wird im vorherigen Beispielcode der Methode gezeigt. `DisplayLogGroupsWithPaginators`

- Synchrone Paginierung

  Synchrone Paginierung ist für Projekte von.NET Framework 4.7.2 (oder höher) verfügbar.

  Um dies zu sehen, erstellen Sie ein.NET Framework 4.7.2 (oder höher) -Projekt und kopieren Sie den vorherigen Code hinein. Entfernen Sie dann einfach das `await` Schlüsselwort aus den beiden `foreach` Paginator-Aufrufen, wie im folgenden Beispiel gezeigt.

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

  Erstellen Sie das Projekt und führen Sie es aus, um dieselben Ergebnisse zu erzielen wie bei der asynchronen Paginierung.

Support für HTTP 2 in der AWS SDK für .NET

Für einige AWS Dienste und Operationen ist HTTP 2 erforderlich. Beispielsweise ist bidirektionales Streaming in Amazon Transcribe Streaming nicht über HTTP 1.1 möglich und erfordert stattdessen HTTP 2. Version 4 der AWS SDK für .NET zusätzlichen Unterstützung für HTTP 2, sodass Sie diese Operationen in Ihren Anwendungen verwenden können. Bei einem bidirektionalen HTTP 2-Vorgang verhält sich das SDK beim Empfang von Streams ähnlich wie bei HTTP 1.1. Das heißt, wenn Anwendungen, die das SDK verwenden, Ereignisse an den Service senden, hat die Anfrage einen Publisher, der vom Entwickler zugewiesen wird.

Sehen Sie sich das folgende Beispiel für Amazon Transcribe Streaming an, um dieses Verhalten in Aktion zu sehen. Es nutzt den Amazon. TranscribeStreamingund Amazon. TranscribeStreaming. Modell-Namespaces.

In diesem Beispiel definiert der Entwickler die `StartStreamTranscriptionRequest.AudioStreamPublisher` Eigenschaft mit einer Callback-Funktion, bei der es sich um eine.NET-Funktion handelt. `Func` Das SDK verwendet das `Func` definierte for`AudioStreamPublisher`, um Ereignisse aus dem Code des Benutzers abzurufen, um sie an den Benutzer zu streamen. Das SDK ruft die auf, `Func` bis es Null zurückgibt.

Der Code zeigt, wie Audio aus einer Datei zur Verarbeitung in Amazon Transcribe Streaming gestreamt werden kann.

```
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**
>
> Einige bidirektionale HTTP 2-Operationen, wie die `InvokeModelWithBidirectionalStreamAsync` Methode von Amazon Bedrock, dem Amazon. BedrockRuntimeNamespace, geben Sie keine Antwort vom ersten Aufruf auf dem Service-Client zurück, bis einige Ereignisse veröffentlicht wurden. Dieses Verhalten könnte dazu führen, dass Ihre Anwendung blockiert wird. Um diese Situation zu vermeiden, trennen Sie den Anwendungscode, der Ereignisse für den Herausgeber bereitstellt, und führen Sie ihn in einem Thread aus, der sich von dem Thread unterscheidet, der Operationen auf dem Service-Client aufruft.

Weitere Überlegungen

- AWS SDK für .NET Unterstützung für HTTP 2 ist nur in Versionen verfügbar, die auf .NET 8 und höher abzielen. Sie ist nicht in Versionen verfügbar, die auf .NET Framework abzielen.
- Ausführlichere Informationen finden Sie unter PR 3730 im aws-sdk-net GitHubRepository.

Zusätzliche Tools

Im Folgenden finden Sie einige zusätzliche Tools, die Sie verwenden können, um die Entwicklung, Bereitstellung und Wartung Ihrer .NET-Anwendungen zu vereinfachen.

AWS Tool bereitstellen

Nachdem Sie Ihre cloudnative.NET Core-Anwendung auf einem Entwicklungscomputer entwickelt haben, können Sie das AWS Deploy Tool für die .NET-CLI verwenden, um Ihre Anwendung einfacher bereitzustellen. AWS

Weitere Informationen finden Sie unter Stellen Sie Anwendungen bereit für AWS.

AWS Framework für die Nachrichtenverarbeitung für.NET

Wenn Sie Dienste wie Amazon SQS, Amazon SNS oder Amazon verwenden EventBridge, können Sie möglicherweise das AWS Message Processing Framework für.NET nutzen. Weitere Informationen finden Sie unter AWS Message Processing Framework für.NET.

Integrationen mit.NET Aspire

Sie können die Vorteile von Integrationen mit.NET Aspire nutzen, um die interne Entwicklungsschleife zu verbessern. Weitere Informationen finden Sie unter Integration AWS mit .NET Aspire in AWS SDK für .NET.

Erweiterte Authentifizierung und Autorisierung mit dem AWS SDK für .NET

Die Themen in diesem Abschnitt enthalten Informationen zu fortgeschrittenen Authentifizierungs- und Autorisierungstechniken in Ihrer AWS SDK für .NET Anwendung.

Themen

- Single Sign-On mit dem AWS SDK für .NET

Single Sign-On mit dem AWS SDK für .NET

AWS IAM Identity Center ist ein cloudbasierter Single Sign-On-Service (SSO), mit dem Sie den SSO-Zugriff auf all Ihre AWS-Konten und Cloud-Anwendungen einfach zentral verwalten können. Vollständige Informationen finden Sie im IAM Identity Center-Benutzerhandbuch.

Wenn Sie nicht wissen, wie ein SDK mit IAM Identity Center interagiert, lesen Sie die folgenden Informationen.

Interaktionsmuster auf hoher Ebene

SDKs Interagieren Sie auf hoher Ebene mit IAM Identity Center auf eine Weise, die dem folgenden Muster ähnelt:

1. IAM Identity Center wird in der Regel über die IAM Identity Center-Konsole konfiguriert, und ein SSO-Benutzer wird zur Teilnahme eingeladen.
2. Die gemeinsam genutzte AWS `config` Datei auf dem Computer des Benutzers wird mit SSO-Informationen aktualisiert.
3. Der Benutzer meldet sich über IAM Identity Center an und erhält kurzfristige Anmeldeinformationen für die AWS Identity and Access Management (IAM-) Berechtigungen, die für ihn konfiguriert wurden. Diese Anmeldung kann über ein Tool, das kein SDK ist, wie z. B. das AWS CLI, oder programmgesteuert über eine .NET-Anwendung initiiert werden.
4. Der Benutzer setzt seine Arbeit fort. Wenn sie andere Anwendungen ausführen, die SSO verwenden, müssen sie sich nicht erneut anmelden, um die Anwendungen zu öffnen.

Der Rest dieses Themas enthält Referenzinformationen zur Einrichtung und Verwendung AWS IAM Identity Center. Es enthält zusätzliche und umfassendere Informationen als die grundlegende SSO-

Einrichtung unterAuthentifizierung mit AWS. Wenn Sie SSO noch nicht kennen AWS, sollten Sie sich zunächst dieses Thema ansehen, um grundlegende Informationen zu erhalten, und sich dann die folgenden Tutorials ansehen, um SSO in Aktion zu erleben:

- Tutorial: Ausschließlich.NET-Anwendung
- Tutorial: AWS CLI und .NET-Anwendung

Dieses Thema enthält die folgenden Abschnitte:

- Voraussetzungen
- Ein SSO-Profil einrichten
- SSO-Token generieren und verwenden
- Weitere Ressourcen
- Tutorials

Voraussetzungen

Bevor Sie IAM Identity Center verwenden können, müssen Sie bestimmte Aufgaben ausführen, z. B. eine Identitätsquelle auswählen und die entsprechenden AWS-Konten Anwendungen konfigurieren. Weitere Informationen finden Sie unter:

- Allgemeine Informationen zu diesen Aufgaben finden Sie unter Erste Schritte im IAM Identity Center-Benutzerhandbuch.
- Konkrete Aufgabenbeispiele finden Sie in der Liste der Tutorials am Ende dieses Themas. Lesen Sie sich jedoch unbedingt die Informationen in diesem Thema durch, bevor Sie die Tutorials ausprobieren.

Ein SSO-Profil einrichten

Nachdem das IAM Identity Center in der entsprechenden Datei konfiguriert wurde AWS-Konto, muss der gemeinsam genutzten AWS `config` Datei des Benutzers ein benanntes Profil für SSO hinzugefügt werden. Dieses Profil wird verwendet, um eine Verbindung zum AWS Zugriffsportal herzustellen, das kurzfristige Anmeldeinformationen für die IAM-Berechtigungen zurückgibt, die für den Benutzer konfiguriert wurden.

Die gemeinsam genutzte `config` Datei wird normalerweise `%USERPROFILE%\.aws\config` in Windows sowie unter Linux und `~/.aws/config` macOS benannt. Sie können Ihren bevorzugten Texteditor verwenden, um ein neues Profil für SSO hinzuzufügen. Alternativ können Sie den `aws configure sso` Befehl verwenden. Weitere Informationen zu diesem Befehl finden Sie unter Konfiguration der AWS CLI für die Verwendung von IAM Identity Center im AWS Command Line Interface Benutzerhandbuch.

Das neue Profil ähnelt dem folgenden:

```
[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
```

Die Einstellungen für das neue Profil sind unten definiert. Die ersten beiden Einstellungen definieren das AWS Zugriffsportal. Bei den anderen beiden Einstellungen handelt es sich um ein Paar, das zusammengenommen die Berechtigungen definiert, die für einen Benutzer konfiguriert wurden. Alle vier Einstellungen sind erforderlich.

`sso_start_url`

Die URL, die auf das AWS Zugriffsportal der Organisation verweist. Um diesen Wert zu finden, öffnen Sie die IAM Identity Center-Konsole, wählen Sie Einstellungen und suchen Sie nach der Portal-URL.

`sso_region`

Der AWS-Region , der den Access-Portal-Host enthält. Dies ist die Region, die bei der Aktivierung von IAM Identity Center ausgewählt wurde. Sie kann sich von den Regionen unterscheiden, die Sie für andere Aufgaben verwenden.

Eine vollständige Liste der AWS-Regionen und ihrer Codes finden Sie unter Regionale Endpunkte in der Allgemeine Amazon Web Services-Referenz.

`sso_account_id`

Die ID eines AWS-Konto , das über den AWS Organizations Dienst hinzugefügt wurde. Um die Liste der verfügbaren Konten zu sehen, gehen Sie zur IAM Identity Center-Konsole und öffnen Sie die AWS-KontenSeite. Die Konto-ID, die Sie für diese Einstellung auswählen, entspricht dem Wert, den Sie der `sso_role_name` Einstellung geben möchten, der als Nächstes angezeigt wird.

`sso_role_name`

Der Name eines IAM Identity Center-Berechtigungssatzes. Dieser Berechtigungssatz definiert die Berechtigungen, die einem Benutzer über IAM Identity Center erteilt werden.

Das folgende Verfahren ist eine Möglichkeit, den Wert für diese Einstellung zu ermitteln.

1. Gehen Sie zur IAM Identity Center-Konsole und öffnen Sie die AWS-KontenSeite.
2. Wählen Sie ein Konto aus, um dessen Details anzuzeigen. Das Konto, das Sie auswählen, enthält den SSO-Benutzer oder die SSO-Gruppe, dem Sie SSO-Berechtigungen erteilen möchten.
3. Sehen Sie sich die Liste der Benutzer und Gruppen an, die dem Konto zugewiesen sind, und suchen Sie den Benutzer oder die Gruppe, für die Sie sich interessieren. Der Berechtigungssatz, den Sie in der `sso_role_name` Einstellung angeben, ist einer der Gruppen, die diesem Benutzer oder dieser Gruppe zugeordnet sind.

Wenn Sie dieser Einstellung einen Wert zuweisen, verwenden Sie den Namen des Berechtigungssatzes, nicht den Amazon-Ressourcennamen (ARN).

Mit den Berechtigungssätzen sind IAM-Richtlinien und Richtlinien für benutzerdefinierte Berechtigungen verknüpft. Weitere Informationen finden Sie unter Berechtigungssätze im IAM Identity Center-Benutzerhandbuch.

SSO-Token generieren und verwenden

Um SSO verwenden zu können, muss ein Benutzer zunächst ein temporäres Token generieren und dieses Token dann verwenden, um auf die entsprechenden AWS Anwendungen und Ressourcen zuzugreifen. Für .NET-Anwendungen können Sie die folgenden Methoden verwenden, um diese temporären Token zu generieren und zu verwenden:

- Erstellen Sie .NET-Anwendungen, die bei Bedarf zuerst ein Token generieren und das Token dann verwenden.
- Generieren Sie ein Token mit dem AWS CLI und verwenden Sie das Token dann in .NET-Anwendungen.

Diese Methoden werden in den folgenden Abschnitten beschrieben und in den Tutorials demonstriert.

> **Important**
>
> Ihre Anwendung muss auf die folgenden NuGet Pakete verweisen, damit die SSO-Auflösung funktioniert:
>
> - `AWSSDK.SSO`
> - `AWSSDK.SSOOIDC`
>
> Wenn diese Pakete nicht referenziert werden, wird eine Laufzeitausnahme ausgelöst.

Ausschließlich.NET-Anwendung

In diesem Abschnitt erfahren Sie, wie Sie eine .NET-Anwendung erstellen, die bei Bedarf ein temporäres SSO-Token generiert und dieses Token dann verwendet. Eine vollständige Anleitung zu diesem Prozess finden Sie unterTutorial für SSO mit ausschließlich.NET-Anwendungen.

Generieren und verwenden Sie programmgesteuert ein SSO-Token

Sie können nicht nur das verwenden AWS CLI, sondern auch programmgesteuert ein SSO-Token generieren.

Zu diesem Zweck erstellt Ihre Anwendung ein `AWSCredentials`Objekt für das SSO-Profil, das temporäre Anmeldeinformationen lädt, sofern welche verfügbar sind. Anschließend muss Ihre Anwendung das Objekt in ein `AWSCredentials` `SSOAWSCredentials`Objekt umwandeln und einige Optionseigenschaften festlegen, einschließlich einer Callback-Methode, mit der der Benutzer bei Bedarf zur Eingabe von Anmeldeinformationen aufgefordert wird.

Diese Methode wird im folgenden Codeausschnitt gezeigt.

> **Important**
>
> Ihre Anwendung muss auf die folgenden NuGet Pakete verweisen, damit die SSO-Auflösung funktioniert:
>
> - `AWSSDK.SSO`
> - `AWSSDK.SSOOIDC`

Wenn diese Pakete nicht referenziert werden, wird eine Laufzeitausnahme ausgelöst.

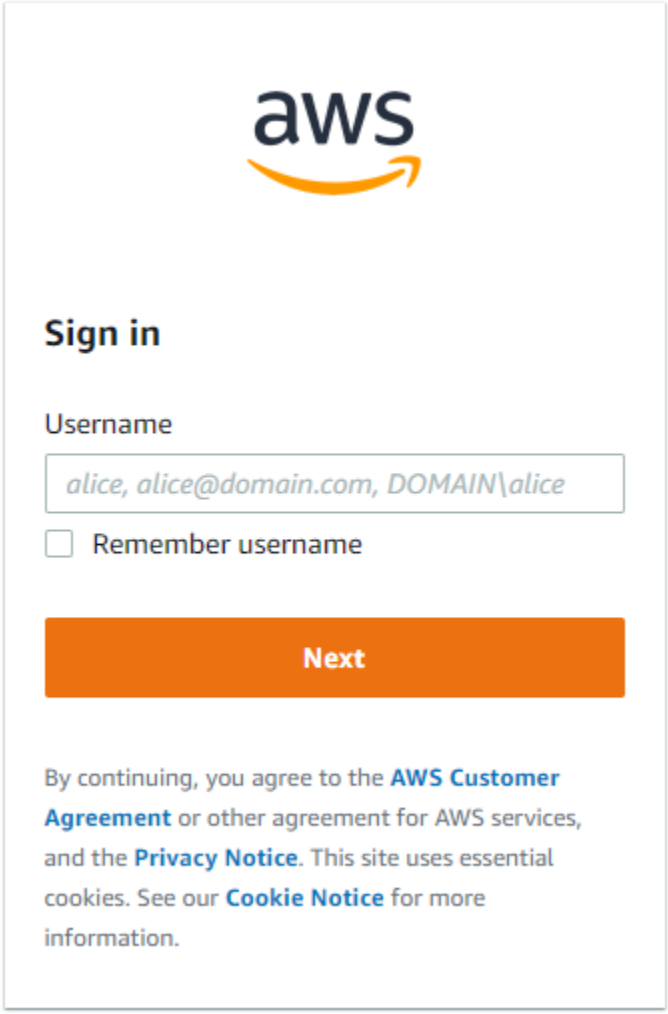
```
static AWSCredentials LoadSsoCredentials()
{
    var chain = new CredentialProfileStoreChain();
    if (!chain.TryGetAWSCredentials("my-sso-profile", out var credentials))
        throw new Exception("Failed to find the my-sso-profile profile");

    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;
}
```

Wenn kein geeignetes SSO-Token verfügbar ist, wird das Standardbrowserfenster geöffnet und die entsprechende Anmeldeseite geöffnet. Wenn Sie beispielsweise IAM Identity Center als Identitätsquelle verwenden, wird dem Benutzer eine Anmeldeseite angezeigt, die der folgenden ähnelt:



> **Note**
>
> Die Textzeichenfolge, die Sie angeben, `SSOAWSCredentials.Options.ClientName` darf keine Leerzeichen enthalten. Wenn die Zeichenfolge Leerzeichen enthält, wird eine Laufzeitausnahme ausgelöst.

[Tutorial für SSO mit ausschließlich.NET-Anwendungen]

AWS CLI und .NET-Anwendung

In diesem Abschnitt erfahren Sie, wie Sie mithilfe von ein temporäres SSO-Token generieren und wie Sie dieses Token in einer Anwendung verwenden. AWS CLI Eine vollständige Anleitung zu diesem Vorgang finden Sie unter[Tutorial für SSO mit den AWS CLI und .NET-Anwendungen].

Generieren Sie ein SSO-Token mit dem AWS CLI

Zusätzlich zur programmgesteuerten Generierung eines temporären SSO-Tokens verwenden Sie den, AWS CLI um das Token zu generieren. Die folgenden Informationen zeigen Ihnen, wie das geht.

Nachdem der Benutzer ein SSO-fähiges Profil erstellt hat, wie in einem vorherigen Abschnitt gezeigt, führt er den `aws sso login` Befehl vom aus aus. AWS CLI Sie müssen sicherstellen, dass der `--profile` Parameter zusammen mit dem Namen des SSO-aktivierten Profils angegeben wird. Das wird im folgenden Beispiel veranschaulicht:

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

Wenn der Benutzer nach Ablauf des aktuellen Tokens ein neues temporäres Token generieren möchte, kann er denselben Befehl erneut ausführen.

Verwenden Sie das generierte SSO-Token in einer .NET-Anwendung

Die folgenden Informationen zeigen Ihnen, wie Sie ein temporäres Token verwenden, das bereits generiert wurde.

> **Important**
>
> Ihre Anwendung muss auf die folgenden NuGet Pakete verweisen, damit die SSO-Auflösung funktioniert:
>
> - `AWSSDK.SSO`
> - `AWSSDK.SSOOIDC`
>
> Wenn diese Pakete nicht referenziert werden, wird eine Laufzeitausnahme ausgelöst.

Ihre Anwendung erstellt ein `AWSCredentials`Objekt für das SSO-Profil, das die temporären Anmeldeinformationen lädt, die zuvor von der generiert wurden AWS CLI. Dies ähnelt den unter gezeigten Methoden Zugreifen auf Anmeldeinformationen und Profile in einer Anwendung und hat die folgende Form:

```
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;
}
```

Das `AWSCredentials` Objekt wird dann an den Konstruktor für einen Service-Client übergeben. Zum Beispiel:

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

> **Note**
>
> Die Verwendung `AWSCredentials` zum Laden temporärer Anmeldeinformationen ist nicht erforderlich, wenn Ihre Anwendung für die Verwendung des `[default]` Profils für SSO erstellt wurde. In diesem Fall kann die Anwendung AWS Dienstclients ohne Parameter erstellen, ähnlich wie "`var client = new AmazonS3Client();`“.

Tutorial für SSO mit den AWS CLI und .NET-Anwendungen

Weitere Ressourcen

Weitere Hilfe finden Sie in den folgenden Ressourcen.

- Was ist IAM Identity Center?
- Konfiguration des AWS CLI für die Verwendung von IAM Identity Center
- Verwenden von IAM Identity Center-Anmeldeinformationen im AWS Toolkit for Visual Studio

Tutorials

Themen

- Tutorial für SSO mit ausschließlich.NET-Anwendungen
- Tutorial für SSO mit den AWS CLI und .NET-Anwendungen

Tutorial für SSO mit ausschließlich.NET-Anwendungen

Dieses Tutorial zeigt Ihnen, wie Sie SSO für eine Basisanwendung und einen SSO-Testbenutzer aktivieren. Es konfiguriert die Anwendung so, dass ein temporäres SSO-Token programmgesteuert generiert wird, anstatt das zu verwenden. AWS CLI

Dieses Tutorial zeigt Ihnen einen kleinen Teil der SSO-Funktionalität in der. AWS SDK für .NET Vollständige Informationen zur Verwendung von IAM Identity Center mit dem AWS SDK für .NET finden Sie im Thema mit Hintergrundinformationen. In diesem Thema finden Sie insbesondere die allgemeine Beschreibung für dieses Szenario im Unterabschnitt. Ausschließlich.NET-Anwendung

> **Note**
>
> Einige der Schritte in diesem Tutorial helfen Ihnen bei der Konfiguration von Diensten wie AWS Organizations IAM Identity Center. Wenn Sie diese Konfiguration bereits durchgeführt haben oder wenn Sie nur an dem Code interessiert sind, können Sie zum Abschnitt mit dem Beispielcode springen.

Voraussetzungen

- Konfigurieren Sie Ihre Entwicklungsumgebung, falls Sie dies noch nicht getan haben. Dies wird in Abschnitten wie Installation und Konfiguration Ihrer Toolchain für die AWS SDK für .NET und beschriebenErste Schritte.
- Identifizieren oder erstellen Sie mindestens eine AWS-Konto , die Sie zum Testen von SSO verwenden können. Für die Zwecke dieses Tutorials wird dieses Konto als Test AWS-Konto - oder einfach als Testkonto bezeichnet.
- Identifizieren Sie einen SSO-Benutzer, der SSO für Sie testen kann. Dies ist eine Person, die SSO und die von Ihnen erstellten Basisanwendungen verwenden wird. Für dieses Tutorial könnten Sie (der Entwickler) oder eine andere Person diese Person sein. Wir empfehlen außerdem ein Setup, bei dem der SSO-Benutzer auf einem Computer arbeitet, der sich nicht in Ihrer Entwicklungsumgebung befindet. Dies ist jedoch nicht unbedingt erforderlich.
- Auf dem Computer des SSO-Benutzers muss ein.NET-Framework installiert sein, das mit dem Framework kompatibel ist, mit dem Sie Ihre Entwicklungsumgebung eingerichtet haben.

Richten Sie ein AWS

In diesem Abschnitt erfahren Sie, wie Sie verschiedene AWS Dienste für dieses Tutorial einrichten.

Um dieses Setup durchzuführen, melden Sie sich zunächst AWS-Konto als Administrator beim Test an. Führen Sie dann die folgenden Schritte aus:

Amazon S3

Gehen Sie zur Amazon S3 S3-Konsole und fügen Sie einige harmlose Buckets hinzu. Später in diesem Tutorial wird der SSO-Benutzer eine Liste dieser Buckets abrufen.

AWS ICH BIN

Gehen Sie zur IAM-Konsole und fügen Sie einige IAM-Benutzer hinzu. Wenn Sie den IAM-Benutzern Berechtigungen erteilen, beschränken Sie die Berechtigungen auf einige harmlose Nur-Lese-Berechtigungen. Später in diesem Tutorial wird der SSO-Benutzer eine Liste dieser IAM-Benutzer abrufen.

AWS Organizations

Gehen Sie zur AWS Organizations Konsole und aktivieren Sie Organizations. Weitere Informationen finden Sie unter Erstellen einer Organisation im Benutzerhandbuch für AWS Organizations .

Diese Aktion fügt der Organisation AWS-Konto den Test als Verwaltungskonto hinzu. Wenn Sie über zusätzliche Testkonten verfügen, können Sie sie einladen, der Organisation beizutreten. Für dieses Tutorial ist dies jedoch nicht erforderlich.

IAM Identity Center

Gehen Sie zur IAM Identity Center-Konsole und aktivieren Sie SSO. Führen Sie bei Bedarf eine E-Mail-Überprüfung durch. Weitere Informationen finden Sie unter Aktivieren von IAM Identity Center im IAM Identity Center-Benutzerhandbuch.

Führen Sie dann die folgende Konfiguration durch.

Konfigurieren Sie IAM Identity Center

1. Gehen Sie zur Seite „Einstellungen“. Suchen Sie nach der „Access-Portal-URL“ und notieren Sie den Wert für die spätere Verwendung in der `sso_start_url` Einstellung.

2. Suchen Sie im Banner von nach dem Wert AWS Management Console AWS-Region , der bei der Aktivierung von SSO festgelegt wurde. Dies ist das Dropdownmenü links neben der AWS-Konto ID. Notieren Sie sich den Regionalcode für die spätere Verwendung in der `sso_region` Einstellung. Dieser Code wird ähnlich sein wie`us-east-1`.
3. Erstellen Sie einen SSO-Benutzer wie folgt:
   a. Gehen Sie zur Benutzerseite.
   b. Wählen Sie Benutzer hinzufügen und geben Sie den Benutzernamen, die E-Mail-Adresse, den Vornamen und den Nachnamen des Benutzers ein. Wählen Sie anschließend Weiter.
   c. Wählen Sie auf der Seite für Gruppen die Option Weiter aus, überprüfen Sie die Informationen und wählen Sie Benutzer hinzufügen.
4. Erstellen Sie eine Gruppe wie folgt:
   a. Gehen Sie zur Seite Gruppen.
   b. Wählen Sie Gruppe erstellen und geben Sie den Gruppennamen und die Beschreibung der Gruppe ein.
   c. Wählen Sie im Abschnitt Benutzer zur Gruppe hinzufügen den SSO-Testbenutzer aus, den Sie zuvor erstellt haben. Wählen Sie dann Gruppe erstellen aus.
5. Erstellen Sie einen Berechtigungssatz wie folgt:
   a. Gehen Sie zur Seite „Berechtigungssätze“ und wählen Sie „Berechtigungssatz erstellen“.
   b. Wählen Sie unter Typ des Berechtigungssatzes die Option Benutzerdefinierter Berechtigungssatz und dann Weiter aus.
   c. Öffnen Sie die Inline-Richtlinie und geben Sie die folgende Richtlinie ein:

      JSON

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

```
            "Resource": "*"
        }
    ]
}
```

d. Geben Sie `SSOReadOnlyRole` für dieses Tutorial den Namen des Berechtigungssatzes ein. Fügen Sie bei Bedarf eine Beschreibung hinzu und wählen Sie dann Weiter.

e. Überprüfen Sie die Informationen und wählen Sie dann Erstellen aus.

f. Notieren Sie sich den Namen des Berechtigungssatzes für die spätere Verwendung in der `sso_role_name` Einstellung.

6. Gehen Sie zur AWS Kontoseite und wählen Sie das AWS Konto aus, das Sie der Organisation zuvor hinzugefügt haben.
7. Suchen Sie im Abschnitt „Übersicht“ auf dieser Seite nach der Konto-ID und notieren Sie sie für die spätere Verwendung in der `sso_account_id` Einstellung.
8. Wählen Sie die Registerkarte Benutzer und Gruppen und dann Benutzer oder Gruppen zuweisen.
9. Wählen Sie auf der Seite Benutzer und Gruppen zuweisen die Registerkarte Gruppen, wählen Sie die Gruppe aus, die Sie zuvor erstellt haben, und klicken Sie auf Weiter.
10. Wählen Sie den zuvor erstellten Berechtigungssatz aus, klicken Sie auf Weiter und anschließend auf Absenden. Die Konfiguration dauert einige Augenblicke.

Erstellen Sie Beispielanwendungen

Erstellen Sie die folgenden Anwendungen. Sie werden auf dem Computer des SSO-Benutzers ausgeführt.

Amazon S3 S3-Buckets auflisten

NuGet Schließt Pakete `AWSSDK.SSO` und `AWSSDK.SSOOIDC` zusätzlich zu `AWSSDK.S3` und `AWSSDK.SecurityToken` ein.

```
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;
        }
    }
}
```

IAM-Benutzer auflisten

NuGet Schließt Pakete `AWSSDK.SSO` und `AWSSDK.SSOOIDC` zusätzlich zu `AWSSDK.IdentityManagement` und `AWSSDK.SecurityToken` ein.

```
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;
        }
    }
}
```

Diese Anwendungen zeigen nicht nur Listen von Amazon S3 S3-Buckets und IAM-Benutzern an, sondern zeigen auch den Benutzeridentitäts-ARN für das SSO-fähige Profil an, das in diesem Tutorial beschrieben wird`my-sso-profile`.

Diese Anwendungen führen SSO-Anmeldeaufgaben durch, indem sie eine Callback-Methode in der Options-Eigenschaft eines Objekts bereitstellen. SSOAWSCredentials

Weisen Sie den SSO-Benutzer an

Bitten Sie den SSO-Benutzer, seine E-Mails zu überprüfen und die SSO-Einladung anzunehmen. Sie werden aufgefordert, ein Passwort festzulegen. Es kann einige Minuten dauern, bis die Nachricht im Posteingang des SSO-Benutzers eingeht.

Geben Sie dem SSO-Benutzer die Anwendungen, die Sie zuvor erstellt haben.

Lassen Sie den SSO-Benutzer dann wie folgt vorgehen:

1. Wenn der Ordner, der die geteilte AWS `config` Datei enthält, nicht existiert, erstellen Sie ihn. Wenn der Ordner existiert und einen Unterordner namens hat`.sso`, löschen Sie diesen Unterordner.

   Der Speicherort dieses Ordners befindet sich normalerweise `%USERPROFILE%\.aws` in Windows sowie `~/.aws` unter Linux und macOS.
2. Erstellen Sie bei Bedarf eine gemeinsam genutzte AWS `config` Datei in diesem Ordner und fügen Sie ihr wie folgt ein Profil hinzu:

   ```
   [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. Führen Sie die Amazon S3 S3-Anwendung aus.
4. Melden Sie sich auf der daraufhin angezeigten Web-Anmeldeseite an. Verwenden Sie den Benutzernamen aus der Einladungsnachricht und das Passwort, das als Antwort auf die Nachricht erstellt wurde.
5. Wenn die Anmeldung abgeschlossen ist, zeigt die Anwendung die Liste der S3-Buckets an.
6. Führen Sie die IAM-Anwendung aus. Die Anwendung zeigt die Liste der IAM-Benutzer an. Dies gilt auch dann, wenn keine zweite Anmeldung durchgeführt wurde. Die IAM-Anwendung verwendet das temporäre Token, das zuvor erstellt wurde.

Bereinigen

Wenn Sie die Ressourcen, die Sie in diesem Tutorial erstellt haben, nicht behalten möchten, bereinigen Sie sie. Dies können AWS Ressourcen oder Ressourcen in Ihrer Entwicklungsumgebung sein, z. B. Dateien und Ordner.

Tutorial für SSO mit den AWS CLI und .NET-Anwendungen

Dieses Tutorial zeigt Ihnen, wie Sie SSO für eine grundlegende .NET-Anwendung und einen SSO-Testbenutzer aktivieren. Es verwendet das AWS CLI , um ein temporäres SSO-Token zu generieren, anstatt es programmgesteuert zu generieren.

Dieses Tutorial zeigt Ihnen einen kleinen Teil der SSO-Funktionalität in der. AWS SDK für .NET Vollständige Informationen zur Verwendung von IAM Identity Center mit dem AWS SDK für .NET finden Sie im Thema mit Hintergrundinformationen. In diesem Thema finden Sie insbesondere die allgemeine Beschreibung für dieses Szenario im Unterabschnitt. AWS CLI und .NET-Anwendung

> **Note**
>
> Einige der Schritte in diesem Tutorial helfen Ihnen bei der Konfiguration von Diensten wie AWS Organizations IAM Identity Center. Wenn Sie diese Konfigurationen bereits durchgeführt haben oder wenn Sie nur an dem Code interessiert sind, können Sie zum Abschnitt mit dem Beispielcode springen.

Voraussetzungen

- Konfigurieren Sie Ihre Entwicklungsumgebung, falls Sie dies noch nicht getan haben. Dies wird in Abschnitten wie Installation und Konfiguration Ihrer Toolchain für die AWS SDK für .NET und beschriebenErste Schritte.
- Identifizieren oder erstellen Sie mindestens eine AWS-Konto , die Sie zum Testen von SSO verwenden können. Für die Zwecke dieses Tutorials wird dieses Konto als Test AWS-Konto - oder einfach als Testkonto bezeichnet.
- Identifizieren Sie einen SSO-Benutzer, der SSO für Sie testen kann. Dies ist eine Person, die SSO und die von Ihnen erstellten Basisanwendungen verwenden wird. Für dieses Tutorial könnten Sie (der Entwickler) oder eine andere Person diese Person sein. Wir empfehlen außerdem ein Setup, bei dem der SSO-Benutzer auf einem Computer arbeitet, der sich nicht in Ihrer Entwicklungsumgebung befindet. Dies ist jedoch nicht unbedingt erforderlich.
- Auf dem Computer des SSO-Benutzers muss ein.NET-Framework installiert sein, das mit dem Framework kompatibel ist, mit dem Sie Ihre Entwicklungsumgebung eingerichtet haben.

- Stellen Sie sicher, dass AWS CLI Version 2 auf dem Computer des SSO-Benutzers installiert ist. Sie können dies überprüfen, indem Sie es `aws --version` in einer Befehlszeile oder einem Terminal ausführen.

Richten Sie ein AWS

In diesem Abschnitt erfahren Sie, wie Sie verschiedene AWS Dienste für dieses Tutorial einrichten.

Um dieses Setup durchzuführen, melden Sie sich zunächst AWS-Konto als Administrator beim Test an. Führen Sie dann die folgenden Schritte aus:

Amazon S3

Gehen Sie zur Amazon S3 S3-Konsole und fügen Sie einige harmlose Buckets hinzu. Später in diesem Tutorial wird der SSO-Benutzer eine Liste dieser Buckets abrufen.

AWS ICH BIN

Gehen Sie zur IAM-Konsole und fügen Sie einige IAM-Benutzer hinzu. Wenn Sie den IAM-Benutzern Berechtigungen erteilen, beschränken Sie die Berechtigungen auf einige harmlose Nur-Lese-Berechtigungen. Später in diesem Tutorial wird der SSO-Benutzer eine Liste dieser IAM-Benutzer abrufen.

AWS Organizations

Gehen Sie zur AWS Organizations Konsole und aktivieren Sie Organizations. Weitere Informationen finden Sie unter Erstellen einer Organisation im Benutzerhandbuch für AWS Organizations .

Diese Aktion fügt der Organisation AWS-Konto den Test als Verwaltungskonto hinzu. Wenn Sie über zusätzliche Testkonten verfügen, können Sie sie einladen, der Organisation beizutreten. Für dieses Tutorial ist dies jedoch nicht erforderlich.

IAM Identity Center

Gehen Sie zur IAM Identity Center-Konsole und aktivieren Sie SSO. Führen Sie bei Bedarf eine E-Mail-Überprüfung durch. Weitere Informationen finden Sie unter Aktivieren von IAM Identity Center im IAM Identity Center-Benutzerhandbuch.

Führen Sie dann die folgende Konfiguration durch.

Konfigurieren Sie IAM Identity Center

1. Gehen Sie zur Seite „Einstellungen“. Suchen Sie nach der „Access-Portal-URL“ und notieren Sie den Wert für die spätere Verwendung in der `sso_start_url` Einstellung.
2. Suchen Sie im Banner von nach dem Wert AWS Management Console AWS-Region , der bei der Aktivierung von SSO festgelegt wurde. Dies ist das Dropdownmenü links neben der AWS-Konto ID. Notieren Sie sich den Regionalcode für die spätere Verwendung in der `sso_region` Einstellung. Dieser Code wird ähnlich sein wie`us-east-1`.
3. Erstellen Sie einen SSO-Benutzer wie folgt:
    a. Gehen Sie zur Benutzerseite.
    b. Wählen Sie Benutzer hinzufügen und geben Sie den Benutzernamen, die E-Mail-Adresse, den Vornamen und den Nachnamen des Benutzers ein. Wählen Sie anschließend Weiter.
    c. Wählen Sie auf der Seite für Gruppen die Option Weiter aus, überprüfen Sie die Informationen und wählen Sie Benutzer hinzufügen.
4. Erstellen Sie eine Gruppe wie folgt:
    a. Gehen Sie zur Seite Gruppen.
    b. Wählen Sie Gruppe erstellen und geben Sie den Gruppennamen und die Beschreibung der Gruppe ein.
    c. Wählen Sie im Abschnitt Benutzer zur Gruppe hinzufügen den SSO-Testbenutzer aus, den Sie zuvor erstellt haben. Wählen Sie dann Gruppe erstellen aus.

5. Erstellen Sie einen Berechtigungssatz wie folgt:

   a. Gehen Sie zur Seite „Berechtigungssätze“ und wählen Sie „Berechtigungssatz erstellen“.

   b. Wählen Sie unter Typ des Berechtigungssatzes die Option Benutzerdefinierter Berechtigungssatz und dann Weiter aus.

   c. Öffnen Sie die Inline-Richtlinie und geben Sie die folgende Richtlinie ein:

      JSON

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

   d. Geben Sie `SSOReadOnlyRole` für dieses Tutorial den Namen des Berechtigungssatzes ein. Fügen Sie bei Bedarf eine Beschreibung hinzu und wählen Sie dann Weiter.

   e. Überprüfen Sie die Informationen und wählen Sie dann Erstellen aus.

   f. Notieren Sie sich den Namen des Berechtigungssatzes für die spätere Verwendung in der `sso_role_name` Einstellung.

6. Gehen Sie zur AWS Kontoseite und wählen Sie das AWS Konto aus, das Sie der Organisation zuvor hinzugefügt haben.

7. Suchen Sie im Abschnitt „Übersicht“ auf dieser Seite nach der Konto-ID und notieren Sie sie für die spätere Verwendung in der `sso_account_id` Einstellung.

8. Wählen Sie die Registerkarte Benutzer und Gruppen und dann Benutzer oder Gruppen zuweisen.

9. Wählen Sie auf der Seite Benutzer und Gruppen zuweisen die Registerkarte Gruppen, wählen Sie die Gruppe aus, die Sie zuvor erstellt haben, und klicken Sie auf Weiter.

10. Wählen Sie den zuvor erstellten Berechtigungssatz aus, klicken Sie auf Weiter und anschließend auf Absenden. Die Konfiguration dauert einige Augenblicke.

Erstellen Sie Beispielanwendungen

Erstellen Sie die folgenden Anwendungen. Sie werden auf dem Computer des SSO-Benutzers ausgeführt.

Amazon S3 S3-Buckets auflisten

NuGet Schließt Pakete `AWSSDK.SSO` und `AWSSDK.SSOOIDC` zusätzlich zu `AWSSDK.S3` und `AWSSDK.SecurityToken` ein.

```
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;
        }
    }
}
```

IAM-Benutzer auflisten

NuGet Schließt Pakete `AWSSDK.SSO` und `AWSSDK.SSOOIDC` zusätzlich zu `AWSSDK.IdentityManagement` und `AWSSDK.SecurityToken` ein.

```
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;
        }
    }
}
```

Diese Anwendungen zeigen nicht nur Listen von Amazon S3 S3-Buckets und IAM-Benutzern an, sondern zeigen auch den Benutzeridentitäts-ARN für das SSO-fähige Profil an, das in diesem Tutorial beschrieben wird`my-sso-profile`.

Weisen Sie den SSO-Benutzer an

Bitten Sie den SSO-Benutzer, seine E-Mails zu überprüfen und die SSO-Einladung anzunehmen. Sie werden aufgefordert, ein Passwort festzulegen. Es kann einige Minuten dauern, bis die Nachricht im Posteingang des SSO-Benutzers eingeht.

Geben Sie dem SSO-Benutzer die Anwendungen, die Sie zuvor erstellt haben.

Lassen Sie den SSO-Benutzer dann wie folgt vorgehen:

1. Wenn der Ordner, der die geteilte AWS `config` Datei enthält, nicht existiert, erstellen Sie ihn. Wenn der Ordner existiert und einen Unterordner namens hat `.sso`, löschen Sie diesen Unterordner.

   Der Speicherort dieses Ordners befindet sich normalerweise `%USERPROFILE%\.aws` in Windows sowie `~/.aws` unter Linux und macOS.

2. Erstellen Sie bei Bedarf eine gemeinsam genutzte AWS `config` Datei in diesem Ordner und fügen Sie ihr wie folgt ein Profil hinzu:

   ```
   [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. Führen Sie die Amazon S3 S3-Anwendung aus. Eine Laufzeitausnahme wird angezeigt.
4. Führen Sie den folgenden AWS CLI Befehl aus:

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

5. Melden Sie sich auf der daraufhin angezeigten Web-Anmeldeseite an. Verwenden Sie den Benutzernamen aus der Einladungsnachricht und das Passwort, das als Antwort auf die Nachricht erstellt wurde.
6. Führen Sie die Amazon S3 S3-Anwendung erneut aus. Die Anwendung zeigt jetzt die Liste der S3-Buckets an.
7. Führen Sie die IAM-Anwendung aus. Die Anwendung zeigt die Liste der IAM-Benutzer an. Dies gilt auch dann, wenn keine zweite Anmeldung durchgeführt wurde. Die IAM-Anwendung verwendet das temporäre Token, das zuvor erstellt wurde.

Bereinigen

Wenn Sie die Ressourcen, die Sie in diesem Tutorial erstellt haben, nicht behalten möchten, bereinigen Sie sie. Dies können AWS Ressourcen oder Ressourcen in Ihrer Entwicklungsumgebung sein, z. B. Dateien und Ordner.

Stellen Sie Anwendungen bereit für AWS

Nachdem Sie Ihre cloudnative.NET Core-Anwendung oder Ihren Dienst auf einem Entwicklungscomputer entwickelt haben, möchten Sie ihn auf AWS diesem Computer bereitstellen. Sie können dies tun, indem Sie die AWS Management Console oder bestimmte Dienste wie AWS CloudFormation oder AWS Cloud Development Kit (AWS CDK) verwenden. Sie können auch AWS Tools verwenden, die für die Bereitstellung erstellt wurden. Mit diesen Tools können Sie Folgendes tun.

Über die .NET-CLI bereitstellen

Sie können die folgenden AWS Tools für.NET CLI verwenden, um Ihre Anwendungen bereitzustellen AWS:

- AWS Bereitstellungstool für.NET CLI — Unterstützt Bereitstellungen auf AWS App RunnerAmazon Elastic Container Service (Amazon ECS) und AWS Elastic Beanstalk.
- AWS Lambda Tools für.NET CLI — Unterstützt die Bereitstellung von AWS Lambda Projekten.

Mit den IDE-Toolkits bereitstellen

Sie können AWS Toolkits verwenden, um Ihre Anwendungen direkt von der IDE Ihrer Wahl aus bereitzustellen:

- AWS Toolkit for Visual Studio

  > **Note**
  >
  > Die Funktion „Veröffentlichen in AWS“ im Toolkit bietet dieselbe Funktionalität wie das AWS Deploy Tool for .NET CLI. Weitere Informationen finden Sie AWS im AWS Toolkit for Visual Studio Benutzerhandbuch unter Veröffentlichen unter.

- AWS Toolkit for JetBrains

  Weitere Informationen finden Sie unter Arbeiten mit AWS serverlosen Anwendungen und Arbeiten mit AWS App Runner.

- AWS Toolkit for VS Code

Weitere Informationen finden Sie unter Arbeiten mit serverlosen Anwendungen und Verwenden. AWS App Runner

- AWS Toolkit for Azure DevOps

Anwendungsfälle

Die folgenden Abschnitte enthalten Anwendungsszenarien für bestimmte Arten von Anwendungen, einschließlich Informationen darüber, wie Sie die .NET-CLI zur Bereitstellung dieser Anwendungen verwenden würden.

- ASP.NET Core-Apps
- .NET-Konsolen-Apps
- Blazor-Apps WebAssembly
- AWS Lambda projekte

ASP.NET Core-Apps

Das AWS Deploy Tool für die .NET-CLI unterstützt Sie bei der Bereitstellung Ihrer ASP.NET-Anwendungen und führt Sie durch den Bereitstellungsprozess. Es ist ein interaktives Tool für die .NET-CLI, das hilft, .NET-Anwendungen mit minimalem AWS Wissen bereitzustellen.

Das Deploy Tool bietet die folgenden Funktionen:

- Berechnungsempfehlungen für Ihre Anwendung — Holen Sie sich die Rechenempfehlungen und erfahren Sie, welche AWS Rechenleistung für Ihre Anwendung am besten geeignet ist.
- Dockerfile-Generierung — Das Tool generiert bei Bedarf ein Dockerfile oder verwendet ein vorhandenes Dockerfile.
- Automatisches Paketieren und Bereitstellen — Das Tool erstellt die Bereitstellungsartefakte, stellt die Infrastruktur mithilfe eines generierten AWS CDK Bereitstellungsprojekts bereit und stellt Ihre Anwendung auf dem ausgewählten Computer bereit. AWS
- Wiederholbare und gemeinsam nutzbare Bereitstellungen — Sie können AWS CDK Bereitstellungsprojekte generieren und ändern, um sie an Ihren spezifischen Anwendungsfall anzupassen. Sie können Ihre Projekte auch versionieren und sie für wiederholbare Bereitstellungen mit Ihrem Team teilen.

- Hilfe beim Lernen AWS CDK für.NET — Das Tool hilft Ihnen dabei, sich schrittweise mit den zugrunde liegenden AWS Tools vertraut zu machen, auf denen es aufbaut, z. B. AWS CDK

Das AWS Deploy Tool unterstützt die Bereitstellung von ASP.NET Core-Anwendungen für die folgenden AWS Dienste:

- Verwendung von Amazon ECS Service AWS Fargate— Unterstützt die Bereitstellung von Webanwendungen auf Amazon Elastic Container Service (Amazon ECS) mit Rechenleistung, die von einer AWS Fargate serverlosen Compute-Engine verwaltet wird.
- AWS App Runner- Unterstützt Bereitstellungen in einem vollständig verwalteten Service, der es Entwicklern erleichtert, containerisierte Webanwendungen in großem Umfang bereitzustellen. APIs Es sind keine Vorkenntnisse in der Infrastruktur erforderlich.
- AWS Elastic Beanstalk- Unterstützt Bereitstellungen in einem Service, der es Entwicklern leicht macht, Webanwendungen bereitzustellen, und APIs in einer vollständig verwalteten Umgebung in großem Umfang. Es sind keine Vorkenntnisse in der Infrastruktur erforderlich.

Weitere Informationen finden Sie in der Tool-Übersicht. Um von dort aus zu beginnen, navigieren Sie zu Dokumentation, Erste Schritte und wählen Sie Installationsanweisungen aus.

.NET-Konsolen-Apps

Das AWS Deploy Tool für die.NET-CLI unterstützt Sie bei der Bereitstellung Ihrer .NET-Konsolenanwendungen als Dienst oder als geplante Aufgabe als Container-Image unter Linux und führt Sie durch den Bereitstellungsprozess. Wenn Ihre Anwendung kein Dockerfile hat, generiert das Tool es automatisch. Andernfalls wird ein vorhandenes Dockerfile verwendet.

Das Deploy Tool verfügt über die folgenden Funktionen:

- Berechnungsempfehlungen für Ihre Anwendung — Holen Sie sich die Rechenempfehlungen und erfahren Sie, welche AWS Rechenleistung für Ihre Anwendung am besten geeignet ist.
- Dockerfile-Generierung — Das Tool generiert bei Bedarf ein Dockerfile oder verwendet ein vorhandenes Dockerfile.
- Automatisches Paketieren und Bereitstellen — Das Tool erstellt die Bereitstellungsartefakte, stellt die Infrastruktur mithilfe eines generierten AWS CDK Bereitstellungsprojekts bereit und stellt Ihre Anwendung auf dem ausgewählten Computer bereit. AWS

- Wiederholbare und gemeinsam nutzbare Bereitstellungen — Sie können AWS CDK Bereitstellungsprojekte generieren und ändern, um sie an Ihren spezifischen Anwendungsfall anzupassen. Sie können Ihre Projekte auch versionieren und sie für wiederholbare Bereitstellungen mit Ihrem Team teilen.
- Hilfe beim Lernen AWS CDK für.NET — Das Tool hilft Ihnen dabei, sich schrittweise mit den zugrunde liegenden AWS Tools vertraut zu machen, auf denen es aufbaut, z. B. AWS CDK

Das AWS Deploy Tool unterstützt die Bereitstellung von.NET-Konsolenanwendungen für die folgenden AWS Dienste:

- Nutzung von Amazon ECS AWS Fargate— Unterstützt Bereitstellungen von.NET-Anwendungen als Service (z. B. als Hintergrundprozessor) für Amazon Elastic Container Service (Amazon ECS) mit Rechenleistung, die von einer AWS Fargate serverlosen Compute Engine verwaltet wird.
- Verwendung von Amazon ECS Scheduled Task AWS Fargate— Unterstützt Bereitstellungen von.NET-Anwendungen als geplante Aufgabe (z. B. end-of-day Prozess) in Amazon ECS mit Rechenleistung, die von einer AWS Fargate serverlosen Compute Engine verwaltet wird.

Weitere Informationen finden Sie in der Tool-Übersicht. Um von dort aus zu beginnen, navigieren Sie zu Dokumentation, Erste Schritte und wählen Sie Installationsanweisungen aus.

Blazor-Apps WebAssembly

Das AWS Deploy Tool für die.NET-CLI hilft Ihnen dabei, Ihre WebAssembly Blazor-Anwendung in Amazon S3 zu hosten und Amazon CloudFront für die Bereitstellung von Inhalten über das Netzwerk zu verwenden. Ihre App wird in einem S3-Bucket für das Webhosting bereitgestellt. Das Tool erstellt und konfiguriert einen S3-Bucket und lädt dann Ihre Blazor-Anwendung in den Bucket hoch.

Das Deploy Tool verfügt über die folgenden Funktionen:

- Automatisches Paketieren und Bereitstellen — Das Tool erstellt die Bereitstellungsartefakte, stellt die Infrastruktur mithilfe eines generierten AWS CDK Bereitstellungsprojekts bereit und stellt Ihre Anwendung auf dem ausgewählten AWS Computer bereit.
- Wiederholbare und gemeinsam nutzbare Bereitstellungen — Sie können AWS CDK Bereitstellungsprojekte generieren und ändern, um sie an Ihren spezifischen Anwendungsfall anzupassen. Sie können Ihre Projekte auch versionieren und sie für wiederholbare Bereitstellungen mit Ihrem Team teilen.

- Hilfe beim Lernen AWS CDK für.NET — Das Tool hilft Ihnen dabei, sich schrittweise mit den zugrunde liegenden AWS Tools vertraut zu machen, auf denen es aufbaut, z. B. AWS CDK

Weitere Informationen finden Sie in der Toolübersicht. Um von dort aus zu beginnen, navigieren Sie zu Dokumentation, Erste Schritte und wählen Sie Installationsanweisungen aus.

AWS Lambda projekte

AWS Lambda ist ein Rechendienst, mit dem Sie Code ausführen können, ohne Server bereitzustellen oder zu verwalten. Er führt Ihren Code auf einer hochverfügbaren Recheninfrastruktur aus und übernimmt die gesamte Verwaltung der Rechenressourcen. Weitere Informationen zu Lambda finden Sie unter Was ist AWS Lambda? im AWS Lambda Entwicklerhandbuch.

Sie können Lambda-Funktionen mithilfe der .NET-Befehlszeilenschnittstelle (CLI) bereitstellen.

Themen

- Voraussetzungen
- Verfügbare Lambda-Befehle
- Schritte zur Bereitstellung

Voraussetzungen

Bevor Sie beginnen, die .NET-CLI zur Bereitstellung von Lambda-Funktionen zu verwenden, müssen Sie die folgenden Voraussetzungen erfüllen:

- Vergewissern Sie sich, dass Sie die .NET-CLI installiert haben. Beispiel: `dotnet --version`. Gehen Sie bei Bedarf zu https://dotnet.microsoft.com/download, um es zu installieren.
- Richten Sie die .NET-CLI so ein, dass sie mit Lambda funktioniert. Eine Beschreibung dazu finden Sie unter.NET Core CLI im AWS Lambda Entwicklerhandbuch. In diesem Verfahren lautet der Bereitstellungsbefehl wie folgt:

  ```
  dotnet lambda deploy-function MyFunction --function-role role
  ```

  Wenn Sie sich nicht sicher sind, wie Sie eine IAM-Rolle für diese Übung erstellen, fügen Sie den `--function-role role` Teil nicht hinzu. Das Tool hilft Ihnen beim Erstellen einer neuen Rolle.

Verfügbare Lambda-Befehle

Um die Lambda-Befehle aufzulisten, die über die .NET-CLI verfügbar sind, öffnen Sie eine Befehlszeile oder ein Terminal und geben Sie ein`dotnet lambda --help`. Die Befehlsausgabe wird der folgenden ähneln:

```
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>
```

Die Ausgabe listet alle Befehle auf, die derzeit verfügbar sind.

Schritte zur Bereitstellung

In den folgenden Anweisungen wird davon ausgegangen, dass Sie ein AWS Lambda .NET-Projekt erstellt haben. Für die Zwecke dieses Verfahrens wird das Projekt benannt`DotNetCoreLambdaTest`.

1. Öffnen Sie eine Befehlszeile oder ein Terminal und navigieren Sie zu dem Ordner, der Ihre .NET-Lambda-Projektdatei enthält.
2. Geben Sie `dotnet lambda deploy-function` ein.
3. Wenn Sie dazu aufgefordert werden, geben Sie die AWS Region ein (die Region, in der Ihre Lambda-Funktion bereitgestellt wird).
4. Wenn Sie dazu aufgefordert werden, geben Sie den Namen der Funktion ein, die bereitgestellt werden soll, `DotNetCoreLambdaTest` z. B. Dies kann der Name einer Funktion sein, die bereits in Ihrer vorhanden ist, AWS-Konto oder einer Funktion, die dort noch nicht bereitgestellt wurde.
5. Wenn Sie dazu aufgefordert werden, wählen oder erstellen Sie die IAM-Rolle, die Lambda bei der Ausführung der Funktion annimmt.

Nach erfolgreichem Abschluss wird die Meldung Neue Lambda-Funktion erstellt angezeigt.

```
Executing publish command
...
(etc.)
New Lambda function created
```

Wenn Sie eine Funktion bereitstellen, die bereits in Ihrem Konto vorhanden ist, fragt die Bereitstellungsfunktion nur nach der AWS Region (falls erforderlich). In diesem Fall endet die Befehlsausgabe mit`Updating code for existing function`.

Nachdem Ihre Lambda-Funktion bereitgestellt wurde, ist sie einsatzbereit. Weitere Informationen finden Sie unter Beispiele für die Verwendung von AWS Lambda.

Lambda überwacht Lambda-Funktionen automatisch für Sie und meldet Metriken über Amazon. CloudWatch Informationen zur Überwachung und Fehlerbehebung Ihrer Lambda-Funktion finden Sie unter Überwachung und Fehlerbehebung bei Lambda-Anwendungen.

Anrufen von AWS Diensten aus dem AWS SDK für .NET

Die folgenden Abschnitte enthalten Beispiele, Tutorials, Aufgaben und Anleitungen, die Ihnen zeigen, wie Sie mit AWS Diensten arbeiten können. AWS SDK für .NET Diese Beispiele und Tutorials basieren auf einer API, die von AWS SDK für .NET bereitgestellt wird. Welche Klassen und Methoden in der API verfügbar sind, finden Sie in der AWS SDK für .NET API-Referenz.

Wenn Sie mit dem noch nicht vertraut sind AWS SDK für .NET, sollten Sie sich zuerst das Erstellen einer einfachen Anwendung Thema ansehen. Es gibt Ihnen eine Einführung in das SDK.

Weitere Codebeispiele finden Sie im AWS Code Examples Repository und im awslabs Repository unter. GitHub

Bevor Sie beginnen, stellen Sie sicher, dass Sie Ihre Umgebung eingerichtet und Ihr Projekt konfiguriert haben. Lesen Sie auch die Informationen unterVerwenden der SDK.

Themen

- Geführte Codebeispiele für den AWS SDK für .NET
- AWS Lambda Für den Rechendienst verwenden
- Hochrangige Bibliotheken und Frameworks für AWS SDK für .NET
- Support für andere AWS Dienste und Konfigurationen

Geführte Codebeispiele für den AWS SDK für .NET

Die folgenden Abschnitte enthalten Codebeispiele und Anleitungen zu den Beispielen. Sie können Ihnen helfen, zu lernen, wie Sie mit AWS Diensten arbeiten können. AWS SDK für .NET Weitere Codebeispiele finden Sie unterCodebeispiele.

> **Note**
>
> Codebeispiele, die spezifisch für Version 3 von AWS SDK für .NET sind ebenfalls verfügbar. Sie finden sie unter Codebeispiele mit Anleitungen und Codebeispielen im AWS SDK für .NET (V3) Developer Guide. Beachten Sie jedoch, dass Sie, wenn Sie die V3-spezifischen Codebeispiele mit Version 4 des SDK (der neuesten Version) verwenden, möglicherweise Anpassungen entsprechend den Informationen unter vornehmen müssen. Migration auf Version 4

Wenn Sie mit dem noch nicht vertraut sind AWS SDK für .NET, sollten Sie sich zuerst das Erstellen einer einfachen Anwendung Thema ansehen. Es gibt Ihnen eine Einführung in das SDK.

Bevor Sie beginnen, stellen Sie sicher, dass Sie Ihre Umgebung eingerichtet und Ihr Projekt konfiguriert haben. Lesen Sie auch die Informationen unterVerwenden der SDK.

Themen

- Zugriff AWS CloudFormation mit dem AWS SDK für .NET
- Benutzer mit Amazon Cognito authentifizieren
- Verwenden von Amazon DynamoDB NoSQL-Datenbanken
- Mit Amazon arbeiten EC2
- Zugriff auf AWS Identity and Access Management (IAM) mit dem AWS SDK für .NET
- Verwenden des Internetspeichers von Amazon Simple Storage Service
- Senden von Benachrichtigungen aus der Cloud mit Amazon Simple Notification Service
- Nachrichtenübermittlung mit Amazon SQS

Zugriff AWS CloudFormation mit dem AWS SDK für .NET

Der AWS SDK für .NET Support AWS CloudFormation, der vorhersagbar und wiederholt AWS Infrastrukturbereitstellungen erstellt und bereitstellt.

APIs

Das AWS SDK für .NET sorgt für Kunden APIs . AWS CloudFormation Sie APIs ermöglichen es Ihnen, mit AWS CloudFormation Funktionen wie Vorlagen und Stacks zu arbeiten. Dieser Abschnitt enthält eine kleine Anzahl von Beispielen, die Ihnen zeigen, welchen Mustern Sie bei der Arbeit mit diesen APIs Modellen folgen können. Den vollständigen Satz von APIs finden Sie in der AWS SDK für .NET API-Referenz (und scrollen Sie zu „Amazon. CloudFormation„).

Die AWS CloudFormation APIs werden von der zur Verfügung gestellt AWSSDK. CloudFormationPaket.

Voraussetzungen

Bevor Sie beginnen, stellen Sie sicher, dass Sie Ihre Umgebung eingerichtet und Ihr Projekt konfiguriert haben. Lesen Sie auch die Informationen unterVerwenden der SDK.

Themen

Themen

- AWS Ressourcen auflisten mit AWS CloudFormation

AWS Ressourcen auflisten mit AWS CloudFormation

Dieses Beispiel zeigt Ihnen, wie Sie mit dem AWS SDK für .NET die Ressourcen AWS CloudFormation stapelweise auflisten können. Das Beispiel verwendet die Low-Level-API. Die Anwendung akzeptiert keine Argumente, sondern sammelt lediglich Informationen für alle Stacks, auf die die Anmeldeinformationen des Benutzers zugreifen können, und zeigt dann Informationen zu diesen Stacks an.

SDK-Referenzen

NuGet Pakete:

- AWSSDK.CloudFormation

Elemente der Programmierung:

- Namespace Amazon. CloudFormation

  Klasse AmazonCloudFormationClient
- Namespace Amazon. CloudFormation.Modell

  Klasse ICloudFormationPaginatorFactory. DescribeStacks

  Klasse DescribeStackResourcesRequest

  Klasse DescribeStackResourcesResponse

  Klassen-Stack

  Klasse StackResource

  Klassen-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;
        }
    }
```

Benutzer mit Amazon Cognito authentifizieren

> **Note**
>
> Die Informationen in diesem Thema beziehen sich speziell auf Projekte, die auf .NET Framework und AWS SDK für .NET Version 3.3 und früher basieren.

Mit Amazon Cognito Identity können Sie eindeutige Identitäten für Ihre Benutzer erstellen und sie für den sicheren Zugriff auf Ihre AWS Ressourcen wie Amazon S3 oder Amazon DynamoDB authentifizieren. Amazon Cognito Identity unterstützt öffentliche Identitätsanbieter wie Amazon, Facebook, Twitter/Digits, Google oder andere OpenID Connect-kompatible Anbieter sowie nicht authentifizierte Identitäten. Amazon Cognito unterstützt auch entwicklerauthentifizierte Identitäten, mit denen Sie Benutzer mithilfe Ihres eigenen Backend-Authentifizierungsprozesses registrieren und authentifizieren können, während Sie weiterhin Amazon Cognito Sync verwenden, um Benutzerdaten zu synchronisieren und auf Ressourcen zuzugreifen. AWS

Weitere Informationen zu Amazon Cognito finden Sie im Amazon Cognito Developer Guide.

Die folgenden Codebeispiele zeigen, wie Sie Amazon Cognito Identity einfach verwenden können. Das Anmeldeinformationsanbieter Beispiel zeigt, wie Benutzeridentitäten erstellt und authentifiziert werden. Das CognitoAuthentication Erweiterungsbibliothek Beispiel zeigt, wie die CognitoAuthentication Erweiterungsbibliothek zur Authentifizierung von Amazon Cognito Cognito-Benutzerpools verwendet wird.

Themen

- Anbieter Amazon Cognito Cognito-Anmeldeinformationen
- Beispiele Amazon CognitoAuthentication Amazon-Erweiterungsbibliotheken

Anbieter Amazon Cognito Cognito-Anmeldeinformationen

> **Note**
>
> Die Informationen in diesem Thema beziehen sich speziell auf Projekte, die auf .NET Framework und AWS SDK für .NET Version 3.3 und früher basieren.

`Amazon.CognitoIdentity.CognitoAWSCredentials`, gefunden in der AWSSDK.CognitoIdentity NuGetpackage, ist ein Anmeldeinformationsobjekt, das Amazon Cognito und the AWS Security Token Service (AWS STS) verwendet, um Anmeldeinformationen für AWS Anrufe abzurufen.

Der erste Schritt beim Einrichten von `CognitoAWSCredentials` besteht darin, einen „Identitätspool“ zu erstellen. (In einem Identitätspool werden Benutzeridentitätsinformationen gespeichert, die speziell für Ihr Konto gelten.) Die Daten können über Client-Plattformen, Geräte und Betriebssysteme hinweg abgerufen werden, sodass die persistenten App-Informationen für einen Benutzer, der Ihre App zunächst auf einem Smartphone und später auf einem Tablet verwendet, noch verfügbar sind. Sie können über die Amazon Cognito Cognito-Konsole einen neuen Identitätspool erstellen. Wenn Sie die Konsole verwenden, erhalten Sie so auch andere Informationen, die Sie benötigen:

- Ihre Kontonummer – Eine 12-stellige Zahl, z. B. 123456789012, die speziell für Ihr Konto gilt.
- ARN der nicht authentifizierten Rolle – Eine Rolle, die von nicht authentifizierten Benutzern übernommen wird. Beispiel: Diese Rolle bietet schreibgeschützten Zugriff auf Ihre Daten.
- ARN der authentifizierten Rolle – Eine Rolle, die von authentifizierten Benutzern übernommen wird. Diese Rolle kann einen umfassenderen Zugriff auf Ihre Daten gewähren.

Cognito einrichten AWSCredentials

Das folgende Codebeispiel zeigt die Einrichtung`CognitoAWSCredentials`, die Sie dann verwenden können, um Amazon S3 als nicht authentifizierter Benutzer aufzurufen. Auf diese Weise können Sie Aufrufe mit nur einer minimale Menge an Daten machen, die für die Authentifizierung des Benutzers erforderlich ist. Da die Benutzerberechtigungen über die Rolle gesteuert werden, können Sie den Zugriff ganz nach Bedarf konfigurieren.

```
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();
}
```

AWS Als nicht authentifizierter Benutzer verwenden

Das folgende Codebeispiel zeigt, wie Sie mit der Verwendung AWS als nicht authentifizierter Benutzer beginnen, sich dann über Facebook authentifizieren und die Anmeldeinformationen aktualisieren können, um Facebook-Anmeldeinformationen zu verwenden. Mit diesem Ansatz können Sie authentifizierten Benutzern über die authentifizierte Rolle unterschiedliche Fähigkeiten erteilen. So ist beispielsweise eine Telefonanwendung möglich, die Benutzern die anonyme Ansicht von Inhalten gewährt, ihnen das Posten von Inhalten aber nur gestattet, wenn sie sich mit einem oder mehreren der konfigurierten Anbieter anmelden.

```
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();
}
```

Das `CognitoAWSCredentials`-Objekt bietet sogar noch mehr Funktionalität, wenn Sie es zusammen mit dem `AmazonCognitoSyncClient` verwenden, der Teil des AWS SDK für .NET ist. Wenn Sie sowohl als auch `AmazonCognitoSyncClient` verwenden`CognitoAWSCredentials`, müssen Sie die `IdentityId` Eigenschaften `IdentityPoolId` und nicht angeben, wenn Sie mit dem aufrufen. `AmazonCognitoSyncClient` Diese Eigenschaften werden automatisch aus `CognitoAWSCredentials` ausgefüllt. Dies wird im nächsten Codebeispiel zusammen mit einem Ereignis veranschaulicht, das Sie immer dann benachrichtigt, wenn sich die `IdentityId` für `CognitoAWSCredentials` ändert. In einigen Fällen kann sich die `IdentityId` ändern. So z. B. auch beim Wechsel von einem nicht authentifizierten zu einem authentifizierten Benutzer.

```
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 = "..."
    });
}
```

Beispiele Amazon CognitoAuthentication Amazon-Erweiterungsbibliotheken

> **Note**
>
> Die Informationen in diesem Thema beziehen sich speziell auf Projekte, die auf .NET Framework und AWS SDK für .NET Version 3.3 und früher basieren.

Die CognitoAuthentication Erweiterungsbibliothek befindet sich in Amazon.Extensions.CognitoAuthentication NuGet Paket, vereinfacht den Authentifizierungsprozess von Amazon Cognito Cognito-Benutzerpools für .NET Core- und Xamarin-Entwickler. Die Bibliothek baut auf der Amazon Cognito Identity Provider-API auf, um API-Aufrufe zur Benutzerauthentifizierung zu erstellen und zu senden.

Verwenden der CognitoAuthentication Erweiterungsbibliothek

Amazon Cognito verfügt über einige integrierte `AuthFlow` `ChallengeName` Werte für einen Standardauthentifizierungsablauf zur Validierung von Benutzername und Passwort über das Secure Remote Password (SRP). Weitere Informationen zum Authentifizierungsablauf finden Sie unter Amazon Cognito Ablauf der Authentifizierung in Benutzerpools.

Die folgenden Beispiele erfordern diese `using`-Anweisungen:

```
// 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;
```

Verwenden Sie die Standardauthentifizierung

Erstellen Sie eine AmazonCognitoIdentityProviderClientmit Anonym AWSCredentials, für die keine signierten Anfragen erforderlich sind. Sie müssen keine Region angeben. Der zugrunde liegende Code ruft `FallbackRegionFactory.GetRegionEndpoint()` auf, wenn keine Region angegeben ist. Erstellen Sie die Objekte `CognitoUserPool` und `CognitoUser`. Rufen Sie die

Methode `StartWithSrpAuthAsync` mit einer `InitiateSrpAuthRequest` auf, in der das Benutzerpasswort enthalten ist.

```
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;

}
```

Authentifizieren Sie sich bei Herausforderungen

Es ist auch einfacher, den Authentifizierungsfluss bei NewPasswordRequired Herausforderungen wie Multi-Factor Authentication (MFA) fortzusetzen. Die einzigen Anforderungen sind die CognitoAuthentication Objekte, das Benutzerkennwort für SRP und die erforderlichen Informationen für die nächste Herausforderung. Diese Informationen werden abgerufen, nachdem der Benutzer zur Eingabe aufgefordert wurde. Der folgende Code zeigt eine Möglichkeit, den Challenge-Typ zu überprüfen und die entsprechenden Antworten für MFA und NewPasswordRequired Challenges während des Authentifizierungsprozesses zu erhalten.

Führen Sie wie zuvor eine grundlegende Authentifizierung durch und geben Sie dann `await` für eine `AuthFlowResponse` aus. Wenn die Antwort empfangen wird, durchlaufen Sie das zurückgegebene `AuthenticationResult`-Objekt. Wenn der `ChallengeName`-Typ `NEW_PASSWORD_REQUIRED` ist, rufen Sie die Methode `RespondToNewPasswordRequiredAsync` auf.

```
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.");
    }

}
```

Verwenden Sie AWS Ressourcen nach der Authentifizierung

Sobald ein Benutzer mithilfe der CognitoAuthentication Bibliothek authentifiziert wurde, besteht der nächste Schritt darin, dem Benutzer den Zugriff auf die entsprechenden AWS Ressourcen zu ermöglichen. Dazu müssen Sie über die Amazon Cognito Federated Identities-Konsole einen Identitätspool erstellen. Indem Sie den Amazon Cognito Cognito-Benutzerpool, den Sie als Anbieter erstellt haben, mithilfe seiner PoolID und ClientID angeben, können Sie den Benutzern Ihres Amazon Cognito Cognito-Benutzerpools den Zugriff auf AWS Ressourcen ermöglichen, die mit Ihrem Konto verbunden sind. Sie können auch unterschiedliche Rollen für authentifizierte und nicht authentifizierte Benutzer angeben, um so Zugriff auf unterschiedliche Ressourcen zu ermöglichen. Sie können diese Regeln in der IAM-Konsole ändern. Hier können Sie im Feld Aktion der an die Richtlinie angefügten Rolle Berechtigungen hinzufügen oder entfernen. Anschließend können Sie mithilfe des entsprechenden Identitätspools, Benutzerpools und Amazon Cognito Cognito-Benutzerinformationen Aufrufe an verschiedene AWS Ressourcen tätigen. Das folgende Beispiel zeigt einen mit SRP authentifizierten Benutzer, der auf die verschiedenen Amazon S3 S3-Buckets zugreift, die durch die Rolle des zugehörigen Identitätspools zugelassen sind.

```
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);
        }
    }
}
```

Weitere Authentifizierungsoptionen

Zusätzlich zu SRP NewPasswordRequired, und MFA bietet die CognitoAuthentication Erweiterungsbibliothek einen einfacheren Authentifizierungsablauf für:

- Custom – Initiiert durch einen Aufruf an `StartWithCustomAuthAsync(InitiateCustomAuthRequest customRequest)`
- RefreshToken - Initiieren Sie mit einem Anruf an `StartWithRefreshTokenAuthAsync(InitiateRefreshTokenAuthRequest refreshTokenRequest)`
- RefreshTokenSRP — Initiieren Sie mit einem Anruf an `StartWithRefreshTokenAuthAsync(InitiateRefreshTokenAuthRequest refreshTokenRequest)`
- AdminNoSRP — Initiieren Sie mit einem Anruf an `StartWithAdminNoSrpAuthAsync(InitiateAdminNoSrpAuthRequest adminAuthRequest)`

Rufen Sie abhängig vom gewünschten Ablauf die entsprechende Methode auf. Fahren Sie dann mit Eingabeaufforderungen an den Benutzer fort, die ihm in den `AuthFlowResponse`-

Objekten eines jeden Methodenaufrufs angezeigt werden. Rufen Sie außerdem die entsprechende Antwortmethode, wie z. B. `RespondToSmsMfaAuthAsync` für MFA-Aufforderungen und `RespondToCustomAuthAsync` für benutzerdefinierte Aufforderungen, auf.

Verwenden von Amazon DynamoDB NoSQL-Datenbanken

> **Note**
>
> Die Programmiermodelle in diesen Themen sind sowohl in.NET Framework als auch in .NET (Core) vorhanden, aber die Aufrufkonventionen unterscheiden sich, unabhängig davon, ob sie synchron oder asynchron sind.

Der AWS SDK für .NET unterstützt Amazon DynamoDB, einen schnellen NoSQL-Datenbankservice, der von angeboten wird. AWS Das SDK bietet drei Programmiermodelle für die Kommunikation mit DynamoDB: das Low-Level-Modell, das Dokumentenmodell und das Objektpersistenzmodell.

Die folgenden Informationen stellen diese und ihre Modelle vor APIs, enthalten Beispiele dafür, wie und wann sie zu verwenden sind, und enthalten Links zu zusätzlichen DynamoDB-Programmierressourcen in der. AWS SDK für .NET

Low-Level-Modell

Das Low-Level-Programmiermodell umschließt direkte Aufrufe des DynamoDB-Dienstes. Sie greifen über den Amazon.Dynamo-Namespace auf dieses Modell zu. DBv2

Von den drei Modellen müssen Sie für das Low-Level-Modell den meisten Code schreiben. Beispielsweise müssen Sie.NET-Datentypen in ihre Entsprechungen in DynamoDB konvertieren. Allerdings bietet Ihnen dieses Modell Zugriff auf die meisten Funktionen.

Die folgenden Beispiele zeigen, wie Sie das Low-Level-Modell verwenden, um eine Tabelle zu erstellen, eine Tabelle zu ändern und Elemente in eine Tabelle in DynamoDB einzufügen.

Erstellen einer Tabelle

Im folgenden Beispiel erstellen Sie eine Tabelle mithilfe der `CreateTable`-Methode der Klasse `AmazonDynamoDBClient`. Die `CreateTable`-Methode verwendet eine Instance der `CreateTableRequest`-Klasse, die Merkmale wie erforderliche Elementattributnamen, Primärschlüsseldefinition und Durchsatzkapazität enthält. Die `CreateTable`-Methode gibt eine Instance der `CreateTableResponse`-Klasse zurück.

```
// 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);
}
```

Überprüfen, ob eine Tabelle für Änderungen bereit ist

Bevor Sie eine Tabelle ändern oder modifizieren können, muss die Tabelle für die Änderung bereit stehen. Das folgende Beispiel zeigt, wie das Low-Level-Modell verwendet wird, um zu überprüfen, ob eine Tabelle in DynamoDB bereit ist. In diesem Beispiel wird die zu überprüfende Zieltabelle durch die `DescribeTable`-Methode der `AmazonDynamoDBClient`-Klasse referenziert. Alle fünf Sekunden überprüft der Code den Wert der `TableStatus`-Eigenschaft der Tabelle. Sobald der Status auf `ACTIVE` gesetzt wird, steht die Tabelle zum Ändern bereit.

```
// 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);
```

Einfügen eines Elements in eine Tabelle

Im folgenden Beispiel verwenden Sie das Low-Level-Modell, um zwei Elemente in eine Tabelle in DynamoDB einzufügen. Jedes Element wird durch die `PutItem`-Methode der `AmazonDynamoDBClient`-Klasse mithilfe einer Instance der `PutItemRequest`-Klasse eingefügt. Jede der beiden Instances der `PutItemRequest`-Klasse akzeptiert den Namen der Tabelle, in die die Elemente eingefügt werden, sowie eine Reihe von Elementattributwerten.

```
// 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);
```

Dokument-Modell

Das Modell der Dokumentenprogrammierung bietet eine einfachere Möglichkeit, mit Daten in DynamoDB zu arbeiten. Dieses Modell ist speziell für den Zugriff auf Tabellen und Elemente in Tabellen vorgesehen. Sie greifen über Amazon.Dynamo auf dieses Modell zu. DBv2 DocumentModelNamespace.

Im Vergleich zum Low-Level-Programmiermodell ist das Dokumentmodell einfacher anhand von DynamoDB-Daten zu codieren. Sie müssen beispielsweise nicht so viele .NET-Datentypen in ihre Entsprechungen in DynamoDB konvertieren. Allerdings bietet dieses Modell auf weniger Funktionen Zugriff als das Low-Level-Programmierungsmodell. Mit diesem Modell können Sie beispielsweise Elemente in Tabellen erstellen, abrufen, aktualisieren und löschen. Zum Erstellen der Tabellen müssen Sie jedoch das Low-Level-Modell verwenden. Im Vergleich zum Object Persistence-Modell müssen Sie bei diesem Modell zum Speichern, Laden und Abfragen von .NET-Objekten mehr Code schreiben.

Weitere Informationen zum DynamoDB-Dokumentprogrammiermodell finden Sie unter.NET: Document Model im Amazon DynamoDB Developer Guide.

Die folgenden Abschnitte enthalten Informationen zum Erstellen einer Darstellung der gewünschten DynamoDB-Tabelle sowie Beispiele zur Verwendung des Dokumentmodells zum Einfügen von Elementen in Tabellen und zum Abrufen von Elementen aus Tabellen.

Erstellen Sie eine Darstellung der Tabelle

Um Datenoperationen mithilfe des Dokumentmodells durchzuführen, müssen Sie zunächst eine Instanz der `Table` Klasse erstellen, die eine bestimmte Tabelle darstellt. Dafür gibt es hauptsächlich zwei Möglichkeiten.

LoadTable Methode

Der erste Mechanismus besteht darin, eine der statischen `LoadTable` Methoden der `Table`Klasse zu verwenden, ähnlich dem folgenden Beispiel:

```
var client = new AmazonDynamoDBClient();
Table table = Table.LoadTable(client, "Reply");
```

> **Note**
>
> Dieser Mechanismus funktioniert zwar, kann aber unter bestimmten Bedingungen aufgrund von Kaltstart- und Threadpool-Verhalten manchmal zu zusätzlicher Latenz oder Deadlocks führen. Weitere Informationen zu diesen Verhaltensweisen finden Sie im Blogbeitrag Verbesserte DynamoDB-Initialisierungsmuster für den. AWS SDK für .NET

TableBuilder

Ein alternativer Mechanismus, die `TableBuilder`Klasse, wurde in Version 3.7.203 des .Dynamo-Pakets eingeführt. AWSSDK DBv2 NuGet Mit diesem Mechanismus können die oben genannten Verhaltensweisen behoben werden, indem bestimmte implizite Methodenaufrufe, insbesondere die Methode, entfernt werden. `DescribeTable` Dieser Mechanismus wird ähnlich wie im folgenden Beispiel verwendet:

```
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();
```

Weitere Informationen zu diesem alternativen Mechanismus finden Sie erneut im Blogbeitrag Verbesserte DynamoDB-Initialisierungsmuster für die. AWS SDK für .NET

Ein Element in eine Tabelle einfügen

Im folgenden Beispiel wird eine Antwort mithilfe der `PutItemAsync` Methode der `Table` Klasse in die Antworttabelle eingefügt. Die `PutItemAsync`-Methode übernimmt eine Instance der `Document`-Klasse, wobei die `Document`-Klasse einfach eine Sammlung von initialisierten Attributen ist.

```
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);
```

Ein Element aus einer Tabelle abrufen

Im folgenden Beispiel wird eine Antwort über die `GetItemAsync` Methode der `Table` Klasse abgerufen. Um zu ermitteln, welche Antwort abgerufen werden soll, verwendet die `GetItemAsync` Methode den hash-and-range Primärschlüssel der Zielantwort.

```
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"]);
```

Im vorherigen Beispiel werden die Tabellenwerte implizit in Zeichenketten für die `WriteLine` Methode konvertiert. Sie können explizite Konvertierungen durchführen, indem Sie die verschiedenen „As [type]“ -Methoden der `DynamoDBEntry` Klasse verwenden. Beispielsweise können Sie den Wert für mit der folgenden `AsGuid()` Methode explizit `Id` von einem `Primitive` Datentyp in eine GUID konvertieren:

```
var guid = reply["Id"].AsGuid();
```

Object Persistence-Modell

Das Objektpersistenz-Programmiermodell wurde speziell für das Speichern, Laden und Abfragen von .NET-Objekten in DynamoDB entwickelt. Sie greifen über Amazon.Dynamo auf dieses Modell zu. DBv2 DataModelNamespace.

Von den drei Modellen ist das Objektpersistenzmodell am einfachsten zu verwenden, wenn Sie DynamoDB-Daten speichern, laden oder abfragen. Sie arbeiten beispielsweise direkt mit DynamoDB-Datentypen. Dieses Modell bietet jedoch nur Zugriff auf Operationen, die .NET-Objekte in DynamoDB speichern, laden und abfragen. Mit diesem Modell können Sie beispielsweise Elemente in Tabellen erstellen, abrufen, aktualisieren und löschen. Sie müssen jedoch zunächst die Tabellen mithilfe des Low-Level-Modells erstellen und dann dieses Modell verwenden, um die .NET-Klassen den Tabellen zuzuordnen.

Weitere Informationen zum DynamoDB-Programmiermodell für Objektpersistenz finden Sie unter.NET: Object persistence model im Amazon DynamoDB Developer Guide.

Die folgenden Beispiele zeigen, wie Sie eine .NET-Klasse definieren, die ein DynamoDB-Element darstellt, eine Instanz der .NET-Klasse verwenden, um ein Element in eine DynamoDB-Tabelle einzufügen, und wie Sie eine Instanz der .NET-Klasse verwenden, um ein Element aus der Tabelle abzurufen.

Definieren einer .NET-Klasse, die ein Element in einer Tabelle darstellt

Im folgenden Beispiel für eine Klassendefinition gibt das `DynamoDBTable` Attribut den Tabellennamen an, während die `DynamoDBRangeKey` Attribute `DynamoDBHashKey` und den hash-and-range Primärschlüssel der Tabelle modellieren. Das `DynamoDBGlobalSecondaryIndexHashKey` Attribut ist so definiert, dass eine Abfrage nach Antworten eines bestimmten Autors erstellt werden kann.

```
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; }
}
```

Erstellen eines Kontextes für das Objektpersistenzmodell

Um das Objektpersistenz-Programmiermodell für DynamoDB zu verwenden, müssen Sie einen Kontext erstellen, der eine Verbindung zu DynamoDB bereitstellt und es Ihnen ermöglicht, auf Tabellen zuzugreifen, verschiedene Operationen durchzuführen und Abfragen auszuführen.

Grundlegender Kontext

Das folgende Beispiel zeigt, wie der einfachste Kontext erstellt wird.

```
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;

var client = new AmazonDynamoDBClient();
var context = new DynamoDBContext(client);
```

Kontext mit DisableFetchingTableMetadata Eigentum

Das folgende Beispiel zeigt, wie Sie zusätzlich die `DisableFetchingTableMetadata` Eigenschaft der `DynamoDBContextConfig` Klasse festlegen können, um implizite Aufrufe der `DescribeTable` Methode zu verhindern.

```
using Amazon.DynamoDBv2;
using Amazon.DynamoDBv2.DataModel;

var client = new AmazonDynamoDBClient();
var context = new DynamoDBContext(client, new DynamoDBContextConfig
{
    DisableFetchingTableMetadata = true
});
```

Wenn die `DisableFetchingTableMetadata` Eigenschaft auf `false` (Standard) gesetzt ist, wie im ersten Beispiel gezeigt, können Sie Attribute, die die Schlüssel- und Indexstruktur von Tabellenelementen beschreiben, aus der `Reply` Klasse weglassen. Diese Attribute werden stattdessen durch einen impliziten Aufruf der Methode abgeleitet. `DescribeTable` Sie `DisableFetchingTableMetadata` ist`true`, wie im zweiten Beispiel gezeigt, auf Methoden des Objektpersistenzmodells gesetzt, die ausschließlich auf den in der Klasse definierten Attributen

`QueryAsync` basieren. `SaveAsync` `Reply` In diesem Fall erfolgt kein Aufruf der `DescribeTable` Methode.

> **Note**
>
> Unter bestimmten Bedingungen können Aufrufe der `DescribeTable` Methode aufgrund von Kaltstart- und Threadpool-Verhalten manchmal zu zusätzlicher Latenz oder Deadlocks führen. Aus diesem Grund ist es manchmal vorteilhaft, Aufrufe dieser Methode zu vermeiden. Weitere Informationen zu diesen Verhaltensweisen finden Sie im Blogbeitrag Verbesserte DynamoDB-Initialisierungsmuster für den. AWS SDK für .NET

Verwenden einer Instanz der .NET-Klasse zum Einfügen eines Elements in eine Tabelle

In diesem Beispiel wird ein Element über die `SaveAsync` Methode der `DynamoDBContext` Klasse eingefügt, die eine initialisierte Instanz der .NET-Klasse verwendet, die das Element darstellt.

```
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"
});
```

Verwenden einer Instanz einer .NET-Klasse, um Elemente aus einer Tabelle abzurufen

In diesem Beispiel wird eine Abfrage erstellt, um alle Datensätze von „Author1" mithilfe der `QueryAsync` Methode der `DynamoDBContext` Klasse zu finden. Anschließend werden die Elemente mit der `GetNextSetAsync` Methode der Abfrage abgerufen.

```
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);
}
```

Zusätzliche Informationen zum Objektpersistenzmodell

Die oben gezeigten Beispiele und Erläuterungen beinhalten manchmal eine Eigenschaft der `DisableFetchingTableMetadata` aufgerufenen `DynamoDBContext` Klasse. Diese Eigenschaft, die in Version 3.7.203 des AWSSDK DBv2 NuGet .Dynamo-Pakets eingeführt wurde, ermöglicht es Ihnen, bestimmte Bedingungen zu vermeiden, die aufgrund von Kaltstart- und Threadpool-Verhalten zu zusätzlichen Latenzen oder Deadlocks führen könnten. Weitere Informationen finden Sie im Blogbeitrag Verbesserte DynamoDB-Initialisierungsmuster für. AWS SDK für .NET

Im Folgenden finden Sie einige zusätzliche Informationen zu dieser Eigenschaft.

- Diese Eigenschaft kann global in Ihrer `app.config` `web.config` OR-Datei festgelegt werden, wenn Sie.NET Framework verwenden.

- Diese Eigenschaft kann mithilfe der `AWSConfigsDynamoDB`Klasse global festgelegt werden, wie im folgenden Beispiel gezeigt.

```
// Set the DisableFetchingTableMetadata property globally
// before constructing any context objects.
AWSConfigsDynamoDB.Context.DisableFetchingTableMetadata = true;

var client = new AmazonDynamoDBClient();
var context = new DynamoDBContext(client);
```

- In einigen Fällen können Sie einer .NET-Klasse keine DynamoDB-Attribute hinzufügen, z. B. wenn die Klasse in einer Abhängigkeit definiert ist. In solchen Fällen ist es dennoch möglich, die Eigenschaft zu nutzen. `DisableFetchingTableMetadata` Verwenden Sie dazu zusätzlich zur `DisableFetchingTableMetadata` Eigenschaft die `TableBuilder`Klasse. Die `TableBuilder` Klasse wurde auch in Version 3.7.203 des AWSSDK DBv2 NuGet .Dynamo-Pakets eingeführt.

```
// 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"
});
```

Weitere Informationen

Verwendung von DynamoDB AWS SDK für .NET zum Programmieren, Informationen und Beispiele

- DynamoDB APIs
- DynamoDB Series Kickoff
- DynamoDB Series - Document Model
- DynamoDB Series - Conversion Schemas
- DynamoDB Series - Object Persistence Model
- DynamoDB Series - Expressions
- Verwenden von Ausdrücken mit Amazon DynamoDB und dem AWS SDK für .NET
- JSON-Unterstützung in Amazon DynamoDB

Low-Level-Modell, Informationen und Beispiele

- Arbeiten mit Tabellen mithilfe der AWS SDK für .NET Low-Level-API
- Mit Elementen arbeiten, die die AWS SDK für .NET Low-Level-API verwenden
- Abfragen von Tabellen mithilfe der Low-Level-API AWS SDK für .NET
- Scannen von Tabellen mithilfe der AWS SDK für .NET Low-Level-API
- Arbeiten mit lokalen Sekundärindizes mithilfe der AWS SDK für .NET Low-Level-API
- Arbeiten mit globalen Sekundärindizes mithilfe der Low-Level-API AWS SDK für .NET

Dokumentenmodell, Informationen und Beispiele

- DynamoDB-Datentypen
- Dynamo DBEntry
- .NET: Dokumentmodell

Modell, Informationen und Beispiele für Objektpersistenz

- .NET: "Object Persistence"-Modell

Weitere nützliche Informationen

- Informationen Integration AWS mit.NET Aspire zur lokalen Entwicklung mit Amazon DynamoDB über.NET Aspire finden Sie unter.

Themen

- Verwenden von Ausdrücken mit Amazon DynamoDB und dem AWS SDK für .NET
- JSON-Unterstützung in Amazon DynamoDB

Verwenden von Ausdrücken mit Amazon DynamoDB und dem AWS SDK für .NET

> **Note**
>
> Die Informationen in diesem Thema beziehen sich speziell auf Projekte, die auf .NET Framework und AWS SDK für .NET Version 3.3 und früher basieren.

Die folgenden Codebeispiele zeigen, wie Sie DynamoDB mit Ausdrücken programmieren. AWS SDK für .NET Ausdrücke bezeichnen die Attribute, die Sie aus einem Element in einer DynamoDB-Tabelle lesen möchten. Sie können auch Ausdrücke beim Schreiben eines Elements verwenden, um alle zu erfüllenden Bedingungen (bedingte Aktualisierung) und die Art, wie die Attribute aktualisiert werden, anzugeben. In einigen Beispielen zur Aktualisierung wird das Attribut durch einen neuen Wert ersetzt oder es werden einer Liste bzw. einer Map neue Daten hinzugefügt. Weitere Informationen finden Sie unter Reading and Writing Items Using Expressions.

Themen

- Beispieldaten
- Abrufen eines einzelnen Elements mithilfe von Ausdrücken und dem Primärschlüssel des Elements
- Abrufen von mehreren Elementen mithilfe von Ausdrücken und dem Primärschlüssel der Tabelle
- Abrufen von mehreren Elementen mithilfe von Ausdrücken und anderen Element-Attributen
- Drucken eines Elements
- Erstellen oder Ersetzen eines Elements mithilfe von Ausdrücken
- Aktualisieren eines Elements mithilfe von Ausdrücken
- Löschen eines Elements mithilfe von Ausdrücken
- Weitere Infos

Beispieldaten

Die Codebeispiele in diesem Thema basieren auf den folgenden beiden Beispielelementen in einer DynamoDB-Tabelle mit dem Namen. `ProductCatalog` Diese Elemente beschreiben Informationen über Produkteinträge im Katalog eines fiktiven Fahrradgeschäftes. Diese Elemente basieren auf dem Beispiel in Case Study: A ProductCatalog Item. Die Datentypbeschreibungen wie `BOOL`, `L`, `M`, `N`, `NS`, `S` und `SS` entsprechen jenen im JSON-Datenformat.

```
{
  "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."
        ]
      }
```

```
    }
  }
}
```

Abrufen eines einzelnen Elements mithilfe von Ausdrücken und dem Primärschlüssel des Elements

Im folgenden Beispiel werden die `Amazon.DynamoDBv2.AmazonDynamoDBClient.GetItem`-Methode sowie eine Gruppe von Ausdrücken zum Abrufen und anschließenden Drucken des Elements mit einer `Id` von `205` veranschaulicht. Es werden nur die folgenden Attribute des Elements zurückgegeben: `Id`, `Title`, `Description`, `Color`, `RelatedItems`, `Pictures` und `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);
```

Im vorangegangenen Beispiel gibt die `ProjectionExpression`-Eigenschaft die zurückzugebenden Attribute an. Die `ExpressionAttributeNames`-Eigenschaft gibt den Platzhalter `#pr` an, der das `ProductReviews`-Attribut repräsentiert sowie den Platzhalter `#ri`, der für das `RelatedItems`-Attribut steht. Das Aufrufen von `PrintItem` bezieht sich auf eine benutzerdefinierte Funktion, wie unter Drucken eines Elements beschrieben.

Abrufen von mehreren Elementen mithilfe von Ausdrücken und dem Primärschlüssel der Tabelle

Im folgenden Beispiel werden die `Amazon.DynamoDBv2.AmazonDynamoDBClient.Query`-Methode sowie eine Gruppe von Ausdrücken zum Abrufen und anschließenden Drucken des Elements mit einer `Id` von `301`, jedoch nur, wenn der Wert von `Price` größer als `150` ist, veranschaulicht. Es werden nur die folgenden Attribute des Elements zurückgegeben: `Id`, `Title` sowie sämtliche `ThreeStar`-Attribute in `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("=====");
}
```

Im vorangegangenen Beispiel gibt die `ProjectionExpression`-Eigenschaft die zurückzugebenden Attribute an. Die `ExpressionAttributeNames`-Eigenschaft gibt den Platzhalter `#pr` an, der das `ProductReviews`-Attribut repräsentiert sowie den Platzhalter `#p`, der für das `Price`-Attribut steht. `#pr.ThreeStar` gibt an, dass nur das `ThreeStar`-Attribut zurückgegeben werden soll. Die `ExpressionAttributeValues`-Eigenschaft gibt den Platzhalter `:val` an, der den Wert `150` repräsentiert. Die `FilterExpression`-Eigenschaft gibt an, dass `#p` (`Price`) größer als `:val` (`150`) sein muss. Das Aufrufen von `PrintItem` bezieht sich auf eine benutzerdefinierte Funktion, wie unter Drucken eines Elements beschrieben.

Abrufen von mehreren Elementen mithilfe von Ausdrücken und anderen Element-Attributen

Im folgenden Beispiel werden die `Amazon.DynamoDBv2.AmazonDynamoDBClient.Scan`-Methode sowie eine Gruppe von Ausdrücken zum Abrufen und anschließenden Drucken aller Elemente mit einer `ProductCategory` von `Bike` veranschaulicht. Es werden nur die folgenden Attribute des Elements zurückgegeben: `Id`, `Title` sowie sämtliche Attribute in `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("=====");
}
```

Im vorangegangenen Beispiel gibt die `ProjectionExpression`-Eigenschaft die zurückzugebenden Attribute an. Die `ExpressionAttributeNames`-Eigenschaft gibt den Platzhalter `#pr` an, der das `ProductReviews`-Attribut repräsentiert sowie den Platzhalter `#pc`, der für das `ProductCategory`-Attribut steht. Die `ExpressionAttributeValues`-Eigenschaft gibt den Platzhalter `:catg` an, der den Wert `Bike` repräsentiert. Die `FilterExpression`-Eigenschaft gibt an, dass `#pc` (`ProductCategory`) gleich `:catg` (`Bike`) sein muss. Das Aufrufen von `PrintItem` bezieht sich auf eine benutzerdefinierte Funktion, wie unter Drucken eines Elements beschrieben.

Drucken eines Elements

Im folgenden Beispiel wird gezeigt, wie die Attribute und Werte eines Elements gedruckt werden. Dieses Beispiel wird in den vorherigen Beispielen verwendet, in denen Folgendes veranschaulicht wurde: Abrufen eines einzelnen Elements mithilfe von Ausdrücken und dem Primärschlüssel des Elements, Abrufen von mehreren Elements mithilfe von Ausdrücken und dem Primärschlüssel der Tabelle sowie Abrufen von mehreren Elements mithilfe von Ausdrücken und anderen Element-Attributen.

```
// 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");
}
```

Im vorherigen Beispiel hat jeder Attributwert mehrere data-type-specific Eigenschaften, die ausgewertet werden können, um das richtige Format für den Ausdruck des Attributs zu bestimmen. Zu diesen Eigenschaften gehören `B`, `BOOL`, `BS`, `L`, `M`, `N`, `NS`, `NULL`, `S` und `SS`, die jenen im JSON-Datenformat entsprechen. Wenn bei Eigenschaften wie `B`, `N`, `NULL` und `S` die entsprechende Eigenschaft nicht `null` ist, besitzt das Attribut den entsprechenden nicht-`null`-Datentyp. Bei Eigenschaften wie`BS`,`L`, `M` `NSSS`, und wenn der Wert größer als Null `Count` ist, hat das Attribut den entsprechenden non-zero-value Datentyp. Wenn alle data-type-specific Eigenschaften des Attributs entweder `null` oder `Count` gleich Null sind, entspricht das Attribut dem `BOOL` Datentyp.

Erstellen oder Ersetzen eines Elements mithilfe von Ausdrücken

Im folgenden Beispiel werden die `Amazon.DynamoDBv2.AmazonDynamoDBClient.PutItem`-Methode sowie eine Gruppe von Ausdrücken zum Aktualisieren des Elements mit einem `Title` von `18-Bicycle 301` veranschaulicht. Wenn das Element noch nicht vorhanden ist, wird ein neues Element hinzugefügt.

```
// 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);
```

Im vorangegangenen Beispiel gibt die `ExpressionAttributeNames`-Eigenschaft den Platzhalter `#title` an, der das `Title`-Attribut repräsentiert. Die `ExpressionAttributeValues`-Eigenschaft gibt den Platzhalter `:product` an, der den Wert `18-Bicycle 301` repräsentiert. Die `ConditionExpression`-Eigenschaft gibt an, dass `#title` (`Title`) gleich `:product` (`18-Bicycle 301`) sein muss. Das Aufrufen von `CreateItemData` bezieht sich auf die folgende benutzerdefinierte Funktion:

```
// 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;
}
```

Im vorherigen Beispiel wird ein Beispielelement mit Beispieldaten an den Aufrufer zurückgegeben. Es werden eine Reihe von Attributen und entsprechenden Werten erstellt, indem Datentypen wie `BOOL`, `L`, `M`, `N`, `NS`, `S` und `SS` verwendet werden, die jenen im JSON-Datenformat entsprechen.

Aktualisieren eines Elements mithilfe von Ausdrücken

Im folgenden Beispiel werden die `Amazon.DynamoDBv2.AmazonDynamoDBClient.UpdateItem`-Methode sowie eine Gruppe von Ausdrücken zum Ändern von `Title` in `18" Girl's Bike` für das Element mit einer `Id` von `301` veranschaulicht.

```
// 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);
```

Im vorangegangenen Beispiel gibt die `ExpressionAttributeNames`-Eigenschaft den Platzhalter `#title` an, der das `Title`-Attribut repräsentiert. Die `ExpressionAttributeValues`-Eigenschaft gibt den Platzhalter `:newproduct` an, der den Wert `18" Girl's Bike` repräsentiert. Die `UpdateExpression`-Eigenschaft gibt an, dass `#title` (`Title`) in `:newproduct` (`18" Girl's Bike`) geändert werden soll.

Löschen eines Elements mithilfe von Ausdrücken

Im folgenden Beispiel werden die `Amazon.DynamoDBv2.AmazonDynamoDBClient.DeleteItem`-Methode sowie eine Gruppe von Ausdrücken zum Löschen eines Elements mit einer `Id` von `301` veranschaulicht, allerdings nur, wenn der `Title` des Elements `18-Bicycle 301` lautet.

```
// 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);
```

Im vorangegangenen Beispiel gibt die `ExpressionAttributeNames`-Eigenschaft den Platzhalter `#title` an, der das `Title`-Attribut repräsentiert. Die `ExpressionAttributeValues`-Eigenschaft gibt den Platzhalter `:product` an, der den Wert `18-Bicycle 301` repräsentiert. Die `ConditionExpression`-Eigenschaft gibt an, dass `#title` (`Title`) gleich `:product` (`18-Bicycle 301`) sein muss.

Weitere Infos

Weitere Informationen und Codeeispiele finden Sie unter

- DynamoDB Series - Expressions
- Angeben von Elementattributen
- Ausdrucksattributnamen
- Bedingungsausdrücke
- Aktualisierungsausdrücke
- Arbeiten mit Elementen mithilfe der AWS SDK für .NET Low-Level-API
- Abfragen von Tabellen mithilfe der Low-Level-API AWS SDK für .NET
- Scannen von Tabellen mithilfe der AWS SDK für .NET Low-Level-API
- Arbeiten mit lokalen Sekundärindizes mithilfe der AWS SDK für .NET Low-Level-API
- Arbeiten mit globalen Sekundärindizes mithilfe der Low-Level-API AWS SDK für .NET

JSON-Unterstützung in Amazon DynamoDB

> **Note**
>
> Die Informationen in diesem Thema beziehen sich speziell auf Projekte, die auf .NET Framework und AWS SDK für .NET Version 3.3 und früher basieren.

Das AWS SDK für .NET unterstützt JSON-Daten bei der Arbeit mit Amazon DynamoDB. Auf diese Weise können Sie einfacher JSON-formatierte Daten aus DynamoDB-Tabellen abrufen und JSON-Dokumente in DynamoDB-Tabellen einfügen.

Themen

- Daten aus einer DynamoDB-Tabelle im JSON-Format abrufen
- Daten im JSON-Format in eine DynamoDB-Tabelle einfügen
- DynamoDB-Datentypkonvertierungen nach JSON
- Weitere Infos

Daten aus einer DynamoDB-Tabelle im JSON-Format abrufen

Das folgende Beispiel zeigt, wie Daten aus einer DynamoDB-Tabelle im JSON-Format abgerufen werden:

```
// 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
//   }
```

Im vorherigen Beispiel wird ein Element aus der Tabelle mithilfe der `ToJson`-Methode der `Document`-Klasse in eine Zeichenfolge im JSON-Format konvertiert. Das Element wird durch die `GetItem`-Methode der `Table`-Klasse abgerufen. Um das abzurufende Element zu bestimmen, verwendet die `GetItem` Methode in diesem Beispiel den hash-and-range Primärschlüssel des Zielelements. Um die Tabelle zu ermitteln, aus der das Element abgerufen werden soll, verwendet die `LoadTable` Methode der `Table` Klasse eine Instanz der `AmazonDynamoDBClient` Klasse und den Namen der Zieltabelle in DynamoDB.

Daten im JSON-Format in eine DynamoDB-Tabelle einfügen

Das folgende Beispiel zeigt, wie das JSON-Format verwendet wird, um ein Element in eine DynamoDB-Tabelle einzufügen:

```
// 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);
```

Im vorherigen Beispiel wird eine Zeichenfolge im JSON-Format mithilfe der `FromJson`-Methode der `Document`-Klasse in ein Element konvertiert. Das Einfügen des Elements in die Tabelle erfolgt mithilfe der `PutItem`-Methode der `Table`-Klasse, die eine Instance der `Document`-Klasse verwendet, die das Element enthält. Um die Tabelle zu bestimmen, in die das Element eingefügt werden soll, wird die `LoadTable` Methode der `Table` Klasse aufgerufen, wobei eine Instanz der `AmazonDynamoDBClient` Klasse und der Name der Zieltabelle in DynamoDB angegeben werden.

DynamoDB-Datentypkonvertierungen nach JSON

Immer wenn Sie die `ToJson` Methode der `Document` Klasse aufrufen und dann für die resultierenden JSON-Daten die `FromJson` Methode aufrufen, um die JSON-Daten wieder in eine Instanz einer `Document` Klasse zu konvertieren, werden einige DynamoDB-Datentypen nicht wie erwartet konvertiert. Das heißt:

- DynamoDB-Sets (die `BS` Typen `SSNS`, und) werden in JSON-Arrays konvertiert.
- DynamoDB-Binärskalare und -mengen (die `BS` Typen `B` und) werden in Base64-kodierte JSON-Zeichenfolgen oder Zeichenkettenlisten konvertiert.

  In diesem Szenario müssen Sie die `DecodeBase64Attributes`-Methode der `Document`-Klasse aufrufen, um die base64-verschlüsselten JSON-Daten durch die korrekte binäre Darstellung zu ersetzen. Im folgenden Beispiel wird ein base64-verschlüsseltes binärskalares Elementattribut in eine Instance einer `Document`-Klasse namens `Picture` durch die korrekte binäre Darstellung ersetzt. In diesem Beispiel wird dasselbe auch für ein base64-verschlüsseltes Binärsatzelementattribut in derselben Instance der `Document`-Klasse namens `RelatedPictures` ausgeführt:

```
item.DecodeBase64Attributes("Picture", "RelatedPictures");
```

Weitere Infos

Weitere Informationen und Beispiele für die Programmierung von JSON mit DynamoDB mit dem finden Sie unter AWS SDK für .NET:

- JSON-Unterstützung in DynamoDB
- Amazon DynamoDB-Aktualisierungen - JSON, Erweitertes kostenloses Kontingent, Flexible Skalierung, Größere Elemente

Mit Amazon arbeiten EC2

Das AWS SDK für .NET unterstützt Amazon EC2, einen Webservice, der skalierbare Rechenkapazität bietet. Sie verwenden diese Rechenkapazität, um Ihre Softwaresysteme zu erstellen und zu hosten.

APIs

Das AWS SDK für .NET bietet APIs EC2 Amazon-Kunden. Sie APIs ermöglichen es Ihnen, mit EC2 Funktionen wie Sicherheitsgruppen und Schlüsselpaaren zu arbeiten. Sie ermöglichen Ihnen APIs auch die Steuerung von EC2 Amazon-Instances. Dieser Abschnitt enthält eine kleine Anzahl von Beispielen, die Ihnen zeigen, welchen Mustern Sie bei der Arbeit mit diesen Modellen folgen können APIs. Den vollständigen Satz von APIs finden Sie in der AWS SDK für .NET API-Referenz (und scrollen Sie zu „Amazon. EC2„).

Die EC2 APIs Amazonen werden von der bereitgestellt AWSSDK. EC2 NuGet Paket.

Voraussetzungen

Bevor Sie beginnen, stellen Sie sicher, dass Sie Ihre Umgebung eingerichtet und Ihr Projekt konfiguriert haben. Lesen Sie auch die Informationen unterVerwenden der SDK.

Informationen zu den Beispielen

Die Beispiele in diesem Abschnitt zeigen Ihnen, wie Sie mit EC2 Amazon-Clients arbeiten und EC2 Amazon-Instances verwalten.

Das EC2 Spot-Instance-Tutorial zeigt Ihnen, wie Sie Amazon EC2 Spot-Instances anfordern. Mit Spot-Instances können Sie für weniger als den On-Demand-Preis auf ungenutzte EC2 Kapazität zugreifen.

Themen

- Arbeiten mit Sicherheitsgruppen in Amazon EC2
- Arbeiten mit EC2 Amazon-Schlüsselpaaren
- Ihre EC2 Amazon-Regionen und Availability Zones anzeigen
- Mit EC2 Amazon-Instances arbeiten
- Anleitung zur Amazon EC2 Spot-Instance

Arbeiten mit Sicherheitsgruppen in Amazon EC2

Bei Amazon EC2 fungiert eine Sicherheitsgruppe als virtuelle Firewall, die den Netzwerkverkehr für eine oder mehrere EC2 Instances steuert. Ordnet Ihre EC2 Instances standardmäßig einer Sicherheitsgruppe zu, die keinen eingehenden Datenverkehr zulässt. Sie können eine Sicherheitsgruppe erstellen, die es Ihren EC2 Instances ermöglicht, bestimmten Datenverkehr zu akzeptieren. Wenn Sie beispielsweise eine Verbindung zu einer EC2 Windows-Instanz herstellen müssen, müssen Sie die Sicherheitsgruppe so konfigurieren, dass RDP-Verkehr zugelassen wird.

Weitere Informationen zu Sicherheitsgruppen finden Sie unter EC2 Amazon-Sicherheitsgruppen im EC2 Amazon-Benutzerhandbuch.

> **Warning**
>
> EC2-Classic wurde am 15. August 2022 eingestellt. Wir empfehlen Ihnen, von EC2 -Classic zu einer VPC zu migrieren. Weitere Informationen finden Sie im Blogbeitrag EC2-Classic Networking is Retiring — So bereiten Sie sich vor.

Informationen zu den Voraussetzungen APIs und Voraussetzungen finden Sie im übergeordneten Abschnitt ()Mit Amazon arbeiten EC2.

Themen

- Auflisten von Sicherheitsgruppen
- Erstellen von Sicherheitsgruppen

- Sicherheitsgruppen werden aktualisiert

Auflisten von Sicherheitsgruppen

Dieses Beispiel zeigt Ihnen, wie Sie die verwenden, AWS SDK für .NET um Sicherheitsgruppen aufzulisten. Wenn Sie eine Amazon Virtual Private Cloud Cloud-ID angeben, listet die Anwendung die Sicherheitsgruppen für diese bestimmte VPC auf. Andernfalls zeigt die Anwendung einfach eine Liste aller verfügbaren Sicherheitsgruppen an.

Die folgenden Abschnitte enthalten Auszüge aus diesem Beispiel. Der vollständige Code für das Beispiel wird danach angezeigt und kann unverändert erstellt und ausgeführt werden.

Themen

- Zählen Sie Sicherheitsgruppen auf
- Vollständiger Code
- Weitere Überlegungen

Zählen Sie Sicherheitsgruppen auf

Der folgende Ausschnitt listet Ihre Sicherheitsgruppen auf. Es listet alle Gruppen oder die Gruppen für eine bestimmte VPC auf, falls eine angegeben ist.

Das Beispiel am Ende dieses Themas zeigt, wie dieser Codeausschnitt verwendet wird.

```
//
// 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();
      }
    }
```

Vollständiger Code

Dieser Abschnitt enthält relevante Referenzen und den vollständigen Code für dieses Beispiel.

SDK-Referenzen

NuGet Pakete:

- AWSSDK.EC2

Elemente der Programmierung:

- Namespace Amazon. EC2

  Klasse EC2Amazon-Kunde
- Namespace Amazon. EC2.Modell

  Klasse DescribeSecurityGroupsRequest

  Klasse DescribeSecurityGroupsResponse

  Klassenfilter

  Klasse SecurityGroup

Der Kodex

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

Weitere Überlegungen

- Beachten Sie für den VPC-Fall, dass der Filter so konstruiert ist, dass der `Name` Teil des Name-Wert-Paares auf „vpc-id“ gesetzt ist. Dieser Name stammt aus der Beschreibung der Eigenschaft der `Filters` Klasse. DescribeSecurityGroupsRequest

- Um die vollständige Liste Ihrer Sicherheitsgruppen zu erhalten, können Sie diese auch DescribeSecurityGroupsAsync ohne Parameter verwenden.

- Sie können die Ergebnisse überprüfen, indem Sie die Liste der Sicherheitsgruppen in der EC2 Amazon-Konsole überprüfen.

Erstellen von Sicherheitsgruppen

Dieses Beispiel zeigt Ihnen, wie Sie AWS SDK für .NET mit dem eine Sicherheitsgruppe erstellen. Sie können die ID einer vorhandenen VPC angeben, für die Sie eine Sicherheitsgruppe EC2 in einer VPC erstellen möchten. Wenn Sie keine solche ID angeben, gilt die neue Sicherheitsgruppe für EC2 -Classic, sofern Ihr AWS Konto dies unterstützt.

Wenn Sie keine VPC-ID angeben und Ihr AWS Konto EC2 -Classic nicht unterstützt, gehört die neue Sicherheitsgruppe zur Standard-VPC Ihres Kontos.

> ⚠ Warning
>
> EC2-Classic wurde am 15. August 2022 eingestellt. Wir empfehlen Ihnen, von EC2 -Classic zu einer VPC zu migrieren. Weitere Informationen finden Sie im Blogbeitrag EC2-Classic Networking is Retiring — So bereiten Sie sich vor.

Die folgenden Abschnitte enthalten Auszüge aus diesem Beispiel. Der vollständige Code für das Beispiel wird danach angezeigt und kann unverändert erstellt und ausgeführt werden.

Themen

- Suchen Sie nach vorhandenen Sicherheitsgruppen
- Eine Sicherheitsgruppe erstellen
- Vollständiger Code

Suchen Sie nach vorhandenen Sicherheitsgruppen

Das folgende Snippet sucht nach vorhandenen Sicherheitsgruppen mit dem angegebenen Namen in der angegebenen VPC.

Das Beispiel am Ende dieses Themas zeigt, wie dieses Snippet verwendet wird.

```
    //
    // 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;
    }
```

Eine Sicherheitsgruppe erstellen

Das folgende Snippet erstellt eine neue Sicherheitsgruppe, wenn eine Gruppe mit diesem Namen in der angegebenen VPC nicht existiert. Wenn keine VPC angegeben ist und eine oder mehrere Gruppen mit diesem Namen existieren, gibt das Snippet einfach die Liste der Gruppen zurück.

Das Beispiel am Ende dieses Themas zeigt, wie dieses Snippet verwendet wird.

```
    //
    // 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;
  }
```

Vollständiger Code

Dieser Abschnitt enthält relevante Referenzen und den vollständigen Code für dieses Beispiel.

SDK-Referenzen

NuGet Pakete:

- AWSSDK.EC2

Elemente der Programmierung:

- Namespace Amazon. EC2

  Klasse EC2Amazon-Kunde
- Namespace Amazon. EC2.Modell

Klasse CreateSecurityGroupRequest

Klasse CreateSecurityGroupResponse

Klasse DescribeSecurityGroupsRequest

Klasse DescribeSecurityGroupsResponse

Klassenfilter

Klasse SecurityGroup

Der Code

```
using System;
using System.Threading.Tasks;
using System.Collections.Generic;
using Amazon.EC2;
using Amazon.EC2.Model;

namespace EC2CreateSecGroup
{
  // = = =
```

```
    // Get the application arguments from the parsed list
    var groupName = CommandLine.GetArgument(parsedArgs, null, "-g", "--group-name");
    var vpcID = CommandLine.GetArgument(parsedArgs, null, "-v", "--vpc-id");
    if(string.IsNullOrEmpty(groupName))
      CommandLine.ErrorExit("\nYou must supply a name for the new group." +
        "\nRun the command with no arguments to see help.");
    if(!string.IsNullOrEmpty(vpcID) && !vpcID.StartsWith("vpc-"))
      CommandLine.ErrorExit($"\nNot a valid VPC ID: {vpcID}");

    // groupName has a value and vpcID either has a value or is null (which is fine)
    // Create the new security group and display information about it
    var securityGroups =
      await CreateSecurityGroup(new AmazonEC2Client(), groupName, vpcID);
    Console.WriteLine("Information about the security group(s):");
    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.");
   }
 }


 // = = =
```

```
        // 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);
    }
  }

}
```

Sicherheitsgruppen werden aktualisiert

Dieses Beispiel zeigt Ihnen, wie Sie AWS SDK für .NET mit dem einer Sicherheitsgruppe eine Regel hinzufügen können. Insbesondere fügt das Beispiel eine Regel hinzu, die eingehenden Datenverkehr an einem bestimmten TCP-Port zulässt, was beispielsweise für Remoteverbindungen zu einer EC2 Instanz verwendet werden kann. Die Anwendung verwendet die ID einer vorhandenen Sicherheitsgruppe, eine IP-Adresse (oder einen Adressbereich) im CIDR-Format und optional eine TCP-Portnummer. Anschließend fügt sie der angegebenen Sicherheitsgruppe eine Regel für eingehenden Datenverkehr hinzu.

> **Note**
>
> Um dieses Beispiel verwenden zu können, benötigen Sie eine IP-Adresse (oder einen Adressbereich) im CIDR-Format. Methoden zum Abrufen der IP-Adresse Ihres lokalen Computers finden Sie unter Zusätzliche Überlegungen am Ende dieses Themas.

Die folgenden Abschnitte enthalten Auszüge aus diesem Beispiel. Der vollständige Code für das Beispiel wird danach angezeigt und kann unverändert erstellt und ausgeführt werden.

Themen

- Fügen Sie eine Regel für eingehenden Datenverkehr hinzu
- Vollständiger Code
- Weitere Überlegungen

Fügen Sie eine Regel für eingehenden Datenverkehr hinzu

Das folgende Snippet fügt einer Sicherheitsgruppe eine Regel für eingehenden Datenverkehr für eine bestimmte IP-Adresse (oder einen bestimmten Bereich) und einen bestimmten TCP-Port hinzu.

Das Beispiel am Ende dieses Themas zeigt, wie dieser Codeausschnitt verwendet wird.

```
    //
    // 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}");
    }
```

Vollständiger Code

Dieser Abschnitt enthält relevante Referenzen und den vollständigen Code für dieses Beispiel.

SDK-Referenzen

NuGet Pakete:

- AWSSDK.EC2

Elemente der Programmierung:

- Namespace Amazon. EC2

  Klasse EC2Amazon-Kunde
- Namespace Amazon. EC2.Modell

Klasse AuthorizeSecurityGroupIngressRequest

Klasse AuthorizeSecurityGroupIngressResponse

Klasse IpPermission

Klasse IpRange

Der Code

```
using System;
using System.Threading.Tasks;
using System.Collections.Generic;
using Amazon.EC2;
using Amazon.EC2.Model;

namespace EC2AddRuleForRDP
{
  // = = =
```

```
      if(string.IsNullOrEmpty(groupID) || !groupID.StartsWith("sg-"))
        CommandLine.ErrorExit("\nThe ID for a security group is missing or
incorrect.");
      if(int.Parse(portStr) == 0)
        CommandLine.ErrorExit($"\nThe given TCP port number, {portStr}, isn't
allowed.");

      // Add a rule to the given security group that allows
      // inbound traffic on a TCP port
      await AddIngressRule(
        new AmazonEC2Client(), groupID, ipAddress, int.Parse(portStr));
    }


    //
    // 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}");
    }


    //
    // 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.");
    }
  }


  // = = =
```

```
        // 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);
    }
  }

}
```

Weitere Überlegungen

- Wenn Sie keine Portnummer angeben, verwendet die Anwendung standardmäßig Port 3389. Dies ist der Port für Windows RDP, über den Sie eine Verbindung zu einer EC2 Instanz herstellen können, auf der Windows ausgeführt wird. Wenn Sie eine EC2 Instance starten, auf der Linux ausgeführt wird, können Sie stattdessen TCP-Port 22 (SSH) verwenden.

- Beachten Sie, dass das Beispiel `IpProtocol` auf „tcp“ gesetzt ist. Die Werte für `IpProtocol` finden Sie in der Beschreibung der `IpProtocol` Eigenschaft der IpPermissionKlasse.

- Wenn Sie dieses Beispiel verwenden, benötigen Sie möglicherweise die IP-Adresse Ihres lokalen Computers. Im Folgenden sind einige Möglichkeiten aufgeführt, wie Sie die Adresse abrufen können.
  - Wenn Ihr lokaler Computer (von dem aus Sie eine Verbindung zu Ihrer EC2 Instance herstellen) über eine statische öffentliche IP-Adresse verfügt, können Sie einen Dienst verwenden, um diese Adresse abzurufen. Ein solcher Dienst ist http://checkip.amazonaws.com/. Weitere Informationen zur Autorisierung von eingehendem Datenverkehr finden Sie unter Regeln zu einer Sicherheitsgruppe hinzufügen und Sicherheitsgruppenregeln für verschiedene Anwendungsfälle im EC2 Amazon-Benutzerhandbuch.
  - Eine andere Möglichkeit, die IP-Adresse Ihres lokalen Computers zu erhalten, ist die Verwendung der EC2 Amazon-Konsole.

    Wählen Sie eine Ihrer Sicherheitsgruppen aus, klicken Sie auf die Registerkarte Regeln für eingehenden Datenverkehr und dann auf Regeln für eingehenden Datenverkehr bearbeiten. Öffnen Sie in einer Regel für eingehenden Datenverkehr das Dropdownmenü in der Spalte Quelle und wählen Sie Meine IP aus, um die IP-Adresse Ihres lokalen Computers im CIDR-Format anzuzeigen. Achten Sie darauf, den Vorgang abzubrechen.

- Sie können die Ergebnisse dieses Beispiels überprüfen, indem Sie die Liste der Sicherheitsgruppen in der EC2Amazon-Konsole überprüfen.

Arbeiten mit EC2 Amazon-Schlüsselpaaren

Amazon EC2 verwendet Public-Key-Kryptografie, um Anmeldeinformationen zu verschlüsseln und zu entschlüsseln. Bei der Kryptografie mit öffentlichen Schlüsseln werden Daten mit einem öffentlichen Schlüssel verschlüsselt, und der Empfänger verwendet dann den privaten Schlüssel, um die Daten zu

entschlüsseln. Der öffentliche und der private Schlüssel werden als Schlüsselpaar bezeichnet. Wenn Sie sich bei einer EC2 Instance anmelden möchten, müssen Sie beim Starten ein key pair angeben und dann den privaten Schlüssel des Paares angeben, wenn Sie eine Verbindung herstellen.

Wenn Sie eine EC2 Instance starten, können Sie ein key pair dafür erstellen oder eines verwenden, das Sie bereits beim Starten anderer Instances verwendet haben. Weitere Informationen zu EC2 Amazon-Schlüsselpaaren finden Sie unter Arbeiten mit EC2 Amazon-Schlüsselpaaren im EC2 Amazon-Benutzerhandbuch.

Informationen zu den Voraussetzungen APIs und Voraussetzungen finden Sie im übergeordneten Abschnitt (Mit Amazon arbeiten EC2).

Themen

- Schlüsselpaare erstellen und anzeigen
- Schlüsselpaare löschen

Schlüsselpaare erstellen und anzeigen

Dieses Beispiel zeigt Ihnen, wie Sie AWS SDK für .NET mit dem ein key pair erstellen. Die Anwendung verwendet den Namen für das neue key pair und den Namen einer PEM-Datei (mit der Erweiterung „.pem“). Sie erstellt das Schlüsselpaar, schreibt den privaten Schlüssel in die PEM-Datei und zeigt dann alle verfügbaren Schlüsselpaare an. Wenn Sie keine Befehlszeilenargumente angeben, zeigt die Anwendung einfach alle verfügbaren Schlüsselpaare an.

Die folgenden Abschnitte enthalten Auszüge aus diesem Beispiel. Der vollständige Code für das Beispiel wird danach angezeigt und kann unverändert erstellt und ausgeführt werden.

Themen

- Erstellen des Schlüsselpaars
- Verfügbare Schlüsselpaare anzeigen
- Vollständiger Code
- Weitere Überlegungen

Erstellen des Schlüsselpaars

Das folgende Snippet erstellt ein key pair und speichert dann den privaten Schlüssel in der angegebenen PEM-Datei.

Das Beispiel am Ende dieses Themas zeigt, wie dieses Snippet verwendet wird.

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

Verfügbare Schlüsselpaare anzeigen

Das folgende Snippet zeigt eine Liste der verfügbaren Schlüsselpaare.

Das Beispiel am Ende dieses Themas zeigt, wie dieses Snippet verwendet wird.

```
//
// 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}");
}
```

Vollständiger Code

Dieser Abschnitt enthält relevante Referenzen und den vollständigen Code für dieses Beispiel.

SDK-Referenzen

NuGet Pakete:

- AWSSDK.EC2

Elemente der Programmierung:

- Namespace Amazon. EC2

  Klasse EC2Amazon-Kunde
- Namespace Amazon. EC2.Modell

  Klasse CreateKeyPairRequest

  Klasse CreateKeyPairResponse

  Klasse DescribeKeyPairsResponse

  Klasse KeyPairInfo

Der Code

```
using System;
using System.Threading.Tasks;
using System.IO;
using Amazon.EC2;
using Amazon.EC2.Model;
using System.Collections.Generic;

namespace EC2CreateKeyPair
{
  // = = =
```

```
    // Parse the command line and show help if necessary
    var parsedArgs = CommandLine.Parse(args);
    if(parsedArgs.Count == 0)
    {
      // In the case of no command-line arguments,
      // just show help and the existing key pairs
      PrintHelp();
      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);
      return;
    }

    // Get the application arguments from the parsed list
    string keyPairName =
      CommandLine.GetArgument(parsedArgs, null, "-k", "--keypair-name");
    string pemFileName =
      CommandLine.GetArgument(parsedArgs, null, "-p", "--pem-filename");
    if(string.IsNullOrEmpty(keyPairName))
      CommandLine.ErrorExit("\nNo key pair name specified." +
        "\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.");
    }
  }


  // = = =
```

```
    //
    // 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);
    }
  }

}
```

Weitere Überlegungen

- Nachdem Sie das Beispiel ausgeführt haben, können Sie das neue key pair in der EC2 Amazon-Konsole sehen.

- Wenn Sie ein key pair erstellen, müssen Sie den zurückgegebenen privaten Schlüssel speichern, da Sie den privaten Schlüssel später nicht abrufen können.

Schlüsselpaare löschen

Dieses Beispiel zeigt Ihnen, wie Sie mit AWS SDK für .NET dem ein key pair löschen können. Die Anwendung verwendet den Namen eines key pair. Es löscht das key pair und zeigt dann alle verfügbaren Schlüsselpaare an. Wenn Sie keine Befehlszeilenargumente angeben, zeigt die Anwendung einfach alle verfügbaren Schlüsselpaare an.

Die folgenden Abschnitte enthalten Auszüge aus diesem Beispiel. Der vollständige Code für das Beispiel wird danach angezeigt und kann unverändert erstellt und ausgeführt werden.

Themen

- Löschen Sie das key pair
- Verfügbare Schlüsselpaare anzeigen
- Vollständiger Code

Löschen Sie das key pair

Das folgende Snippet löscht ein key pair.

Das Beispiel am Ende dieses Themas zeigt, wie dieses Snippet verwendet wird.

```
//
// 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).");
}
```

Verfügbare Schlüsselpaare anzeigen

Das folgende Snippet zeigt eine Liste der verfügbaren Schlüsselpaare.

Das Beispiel am Ende dieses Themas zeigt, wie dieses Snippet verwendet wird.

```
//
// 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}");
}
```

Vollständiger Code

Dieser Abschnitt enthält relevante Referenzen und den vollständigen Code für dieses Beispiel.

SDK-Referenzen

NuGet Pakete:

- AWSSDK.EC2

Elemente der Programmierung:

- Namespace Amazon. EC2

  Klasse EC2Amazon-Kunde
- Namespace Amazon. EC2.Modell

  Klasse DeleteKeyPairRequest

  Klasse DescribeKeyPairsResponse

  Klasse KeyPairInfo

Der Code

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

Ihre EC2 Amazon-Regionen und Availability Zones anzeigen

Amazon EC2 wird an mehreren Standorten weltweit gehostet. Diese Standorte bestehen aus - Regionen und Availability Zones. Jede Region ist ein separates geografisches Gebiet mit mehreren isolierten Standorten, den sogenannten Availability Zones.

Weitere Informationen zu Regionen und Availability Zones finden Sie unter Regionen und Zonen im EC2 Amazon-Benutzerhandbuch.

Dieses Beispiel zeigt Ihnen, wie Sie mithilfe von Details AWS SDK für .NET zu den Regionen und Availability Zones eines EC2 Kunden abrufen können. Die Anwendung zeigt Listen der Regionen und Availability Zones an, die einem EC2 Client zur Verfügung stehen.

SDK-Referenzen

NuGet Pakete:

- AWSSDK.EC2

Elemente der Programmierung:

- Namespace Amazon. EC2

  Klasse EC2Amazon-Kunde
- Namespace Amazon. EC2.Modell

  Klasse DescribeAvailabilityZonesResponse

  Klasse DescribeRegionsResponse

  Klasse AvailabilityZone

  Klasse 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);
    }
  }
}
```

Mit EC2 Amazon-Instances arbeiten

Sie können die verwenden AWS SDK für .NET , um EC2 Amazon-Instances mit Vorgängen wie Erstellen, Starten und Beenden zu steuern. Die Themen in diesem Abschnitt enthalten einige Beispiele dafür, wie Sie dies tun können. Weitere Informationen zu EC2 Instances finden Sie unter EC2 Amazon-Instances im EC2 Amazon-Benutzerhandbuch.

Informationen zu den Voraussetzungen APIs und Voraussetzungen finden Sie im übergeordneten Abschnitt (Mit Amazon arbeiten EC2).

Themen

- Starten einer EC2 Amazon-Instance
- Beenden einer Amazon-Instance EC2

Starten einer EC2 Amazon-Instance

Dieses Beispiel zeigt Ihnen, wie Sie die verwenden AWS SDK für .NET , um eine oder mehrere identisch konfigurierte EC2 Amazon-Instances von demselben Amazon Machine Image (AMI) aus zu starten. Mithilfe mehrerer von Ihnen eingegebener Eingaben startet die Anwendung eine EC2 Instance und überwacht die Instance dann, bis sie den Status „Ausstehend“ erreicht hat.

Wenn Ihre EC2 Instance läuft, können Sie eine Remoteverbindung zu ihr herstellen, wie unter beschrieben(optional) Connect zur Instanz her.

> **Warning**
>
> EC2-Classic wurde am 15. August 2022 eingestellt. Wir empfehlen Ihnen, von EC2 -Classic zu einer VPC zu migrieren. Weitere Informationen finden Sie im Blogbeitrag EC2-Classic Networking is Retiring — So bereiten Sie sich vor.

In den folgenden Abschnitten finden Sie Auszüge und weitere Informationen zu diesem Beispiel. Der vollständige Code für das Beispiel wird hinter den Codefragmenten angezeigt und kann unverändert erstellt und ausgeführt werden.

Themen

- Sammle, was du brauchst

- Starten einer -Instance
- Überwachen Sie die Instanz
- Vollständiger Code
- Weitere Überlegungen
- (optional) Connect zur Instanz her
- Bereinigen

Sammle, was du brauchst

Um eine EC2 Instance zu starten, benötigen Sie mehrere Dinge.

- Eine VPC, auf der die Instance gestartet wird. Wenn es sich um eine Windows-Instance handelt und Sie über RDP eine Verbindung zu ihr herstellen, muss an die VPC höchstwahrscheinlich ein Internet-Gateway sowie ein Eintrag für das Internet-Gateway in der Routing-Tabelle angeschlossen sein. Weitere Informationen finden Sie unter Internet-Gateways im Amazon-VPC-Benutzerhandbuch.

- Die ID eines vorhandenen Subnetzes in der VPC, in dem die Instance gestartet wird. Eine einfache Möglichkeit, dies zu finden oder zu erstellen, besteht darin, sich bei der Amazon VPC-Konsole anzumelden. Sie können es aber auch programmgesteuert abrufen, indem Sie die CreateSubnetAsyncMethoden und verwenden. DescribeSubnetsAsync

  > **Note**
  >
  > Wenn Sie diesen Parameter nicht angeben, wird die neue Instance in der Standard-VPC für Ihr Konto gestartet.

- Die ID einer vorhandenen Sicherheitsgruppe, die zu der VPC gehört, in der die Instance gestartet wird. Weitere Informationen finden Sie unter Arbeiten mit Sicherheitsgruppen in Amazon EC2.

- Wenn Sie eine Verbindung mit der neuen Instanz herstellen möchten, muss die zuvor erwähnte Sicherheitsgruppe über eine entsprechende Regel für eingehenden Datenverkehr verfügen, die SSH-Verkehr auf Port 22 (Linux-Instanz) oder RDP-Verkehr auf Port 3389 (Windows-Instanz) zulässt. Informationen dazu finden Sie unterSicherheitsgruppen werden aktualisiert, auch am Ende dieses Weitere Überlegungen Themas.

- Das Amazon Machine Image (AMI), das zur Erstellung der Instance verwendet wird. Weitere Informationen dazu AMIs finden Sie unter Amazon Machine Images (AMIs) im EC2 Amazon-Benutzerhandbuch. Insbesondere siehe Find an AMI and Shared AMIs.

- Der Name eines vorhandenen EC2 key pair, das für die Verbindung mit der neuen Instanz verwendet wird. Weitere Informationen finden Sie unter Arbeiten mit EC2 Amazon-Schlüsselpaaren.

- Der Name der PEM-Datei, die den privaten Schlüssel des zuvor erwähnten EC2 key pair enthält. Die PEM-Datei wird verwendet, wenn Sie eine Remoteverbindung mit der Instanz herstellen.

Starten einer -Instance

Das folgende Snippet startet eine Instanz. EC2

Das Beispiel am Ende dieses Themas zeigt, wie dieses Snippet verwendet wird.

```
//
// 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;
}
```

Überwachen Sie die Instanz

Das folgende Snippet überwacht die Instanz, bis sie den Status „Ausstehend“ erreicht hat.

Das Beispiel am Ende dieses Themas zeigt, wie dieses Snippet verwendet wird.

Die gültigen Werte der Eigenschaft finden Sie in der InstanceState`Instance.State.Code`Klasse.

```
    //
    // 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}");
      }
    }
```

Vollständiger Code

Dieser Abschnitt enthält relevante Referenzen und den vollständigen Code für dieses Beispiel.

SDK-Referenzen

NuGet Pakete:

- AWSSDK.EC2

Elemente der Programmierung:

- Namespace Amazon. EC2

  Klasse EC2Amazon-Kunde

  Klasse InstanceType
- Namespace Amazon. EC2.Modell

  Klasse DescribeInstancesRequest

  Klasse DescribeInstancesResponse

  Klasseninstanz

  Klasse InstanceNetworkInterfaceSpecification

  Klasse RunInstancesRequest

  Klasse RunInstancesResponse

Der Code

```
using System;
using System.Threading;
using System.Threading.Tasks;
using System.Collections.Generic;
using Amazon.EC2;
using Amazon.EC2.Model;

namespace EC2LaunchInstance
{
  // = = =
```

```
      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);
    }
  }
```

```
}
```

Weitere Überlegungen

- Wenn Sie den Status einer EC2 Instanz überprüfen, können Sie der `Filter` Eigenschaft des DescribeInstancesRequestObjekts einen Filter hinzufügen. Mit dieser Technik können Sie die Anfrage auf bestimmte Instanzen beschränken, z. B. auf Instanzen mit einem bestimmten benutzerdefinierten Tag.

- Der Kürze halber wurden einigen Eigenschaften typische Werte zugewiesen. Einige oder alle dieser Eigenschaften können stattdessen programmgesteuert oder durch Benutzereingaben bestimmt werden.

- Die Werte, die Sie für die `MaxCount` Eigenschaften `MinCount` und des RunInstancesRequestObjekts verwenden können, werden durch die Ziel-Availability Zone und die maximale Anzahl von Instanzen bestimmt, die Sie für den Instance-Typ verwenden können. Weitere Informationen finden Sie unter Wie viele Instances kann ich in Amazon ausführen? EC2 in den EC2 allgemeinen Häufig gestellten Fragen zu Amazon.

- Wenn Sie einen anderen Instance-Typ als dieses Beispiel verwenden möchten, stehen Ihnen mehrere Instance-Typen zur Auswahl. Weitere Informationen finden Sie unter EC2Amazon-Instance-Typen im EC2 Amazon-Benutzerhandbuch. Weitere Informationen finden Sie unter Details zum Instanztyp und Instanztyp-Explorer.

- Sie können einer Instance auch eine IAM-Rolle zuordnen, wenn Sie sie starten. Erstellen Sie dazu ein IamInstanceProfileSpecificationObjekt, dessen `Name` Eigenschaft auf den Namen einer IAM-Rolle gesetzt ist. Fügen Sie dieses Objekt dann der `IamInstanceProfile` Eigenschaft des RunInstancesRequestObjekts hinzu.

  > **Note**
  >
  > Um eine EC2 Instance zu starten, der eine IAM-Rolle zugewiesen ist, muss die Konfiguration eines IAM-Benutzers bestimmte Berechtigungen enthalten. Weitere Informationen zu den erforderlichen Berechtigungen finden Sie im EC2 Amazon-

Benutzerhandbuch unter Gewähren Sie einem Benutzer die Erlaubnis, eine IAM-Rolle an eine Instance zu übergeben.

(optional) Connect zur Instanz her

Nachdem eine Instanz ausgeführt wurde, können Sie mithilfe des entsprechenden Remote-Clients eine Remoteverbindung zu ihr herstellen. Sowohl für Linux- als auch für Windows-Instances benötigen Sie die öffentliche IP-Adresse oder den öffentlichen DNS-Namen der Instanz. Sie benötigen außerdem Folgendes.

Für Linux-Instanzen

Sie können einen SSH-Client verwenden, um eine Verbindung zu Ihrer Linux-Instance herzustellen. Stellen Sie sicher, dass die Sicherheitsgruppe, die Sie beim Start der Instance verwendet haben, SSH-Verkehr auf Port 22 zulässt, wie unter beschrieben. Sicherheitsgruppen werden aktualisiert

Sie benötigen auch den privaten Teil des key pair, das Sie zum Starten der Instance verwendet haben, d. h. die PEM-Datei.

Weitere Informationen finden Sie unter Connect to your Linux Instance im EC2 Amazon-Benutzerhandbuch.

Für Windows-Instances

Sie können einen RDP-Client verwenden, um eine Verbindung zu Ihrer Instance herzustellen. Stellen Sie sicher, dass die Sicherheitsgruppe, die Sie beim Start der Instance verwendet haben, RDP-Verkehr auf Port 3389 zulässt, wie unter beschrieben. Sicherheitsgruppen werden aktualisiert

Sie benötigen außerdem das Administratorkennwort. Sie können dies mithilfe des folgenden Beispielcodes abrufen, der die Instanz-ID und den privaten Teil des key pair benötigt, das zum Starten der Instance verwendet wurde, d. h. die PEM-Datei.

Weitere Informationen finden Sie unter Connect to your Windows Instance im EC2 Amazon-Benutzerhandbuch.

⚠ Warning

Dieser Beispielcode gibt das Administratorkennwort im Klartext für Ihre Instance zurück.

SDK-Referenzen

NuGet Pakete:

- AWSSDK.EC2

Elemente der Programmierung:

- Namespace Amazon. EC2

  Klasse EC2Amazon-Kunde
- Namespace Amazon. EC2.Modell

  Klasse GetPasswordDataRequest

  Klasse GetPasswordDataResponse

Der Code

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Threading.Tasks;
using Amazon.EC2;
using Amazon.EC2.Model;

namespace EC2GetWindowsPassword
{
  // = = =
```

```
      // Get the application arguments from the parsed list
      string instanceID =
        CommandLine.GetArgument(parsedArgs, null, "-i", "--instance-id");
      string pemFileName =
        CommandLine.GetArgument(parsedArgs, null, "-p", "--pem-filename");
      if(   (string.IsNullOrEmpty(instanceID) || !instanceID.StartsWith("i-"))
         || (string.IsNullOrEmpty(pemFileName) || !pemFileName.EndsWith(".pem")))
        CommandLine.ErrorExit(
          "\nOne or more of the required arguments is missing or incorrect." +
          "\nRun the command with no arguments to see help.");

      // Create the EC2 client
      var ec2Client = new AmazonEC2Client();

      // Get and display the password
      string password = await GetPassword(ec2Client, instanceID, pemFileName);
      Console.WriteLine($"\nPassword: {password}");
    }


    //
    // Method to get the administrator password of a Windows EC2 instance
    private static async Task<string> GetPassword(
      IAmazonEC2 ec2Client, string instanceID, string pemFilename)
    {
      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.");
    }
  }

  // = = =
```

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

```
}
```

Bereinigen

Wenn Sie Ihre EC2 Instance nicht mehr benötigen, beenden Sie sie unbedingt, wie unter beschriebenBeenden einer Amazon-Instance EC2 .

Beenden einer Amazon-Instance EC2

Wenn Sie eine oder mehrere Ihrer EC2 Amazon-Instances nicht mehr benötigen, können Sie sie kündigen.

Dieses Beispiel zeigt Ihnen, wie Sie die verwenden, AWS SDK für .NET um EC2 Instances zu beenden. Es verwendet eine Instanz-ID als Eingabe.

SDK-Referenzen

NuGet Pakete:

- AWSSDK.EC2

Elemente der Programmierung:

- Namespace Amazon. EC2

  Klasse EC2Amazon-Kunde
- Namespace Amazon. EC2.Modell

  Klasse TerminateInstancesRequest

  Klasse 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);
      }
    }
  }
}
```

Nachdem Sie das Beispiel ausgeführt haben, empfiehlt es sich, sich bei der EC2 Amazon-Konsole anzumelden, um zu überprüfen, ob die EC2 Instance beendet wurde.

Anleitung zur Amazon EC2 Spot-Instance

Dieses Tutorial zeigt Ihnen, wie Sie Amazon EC2 Spot-Instances verwalten. AWS SDK für .NET

Übersicht

Mit Spot-Instances können Sie ungenutzte EC2 Amazon-Kapazität für weniger als den On-Demand-Preis anfordern. Dadurch können Sie Ihre EC2 Kosten für Anwendungen, die unterbrochen werden können, erheblich senken.

Im Folgenden finden Sie eine allgemeine Zusammenfassung darüber, wie Spot-Instances angefordert und verwendet werden.

1. Erstellen Sie eine Spot-Instance-Anfrage und geben Sie den Höchstpreis an, den Sie zu zahlen bereit sind.
2. Wenn die Anfrage erfüllt ist, führen Sie die Instance wie jede andere EC2 Amazon-Instance aus.
3. Führen Sie die Instance so lange aus, wie Sie möchten, und beenden Sie sie dann, sofern sich der Spot-Preis nicht so ändert, dass die Instance für Sie beendet wird.
4. Bereinigen Sie die Spot-Instance-Anfrage, wenn Sie sie nicht mehr benötigen, sodass keine Spot-Instances mehr erstellt werden.

Dies war ein sehr allgemeiner Überblick über Spot-Instances. Weitere Informationen zu Spot-Instances finden Sie unter Spot-Instances im EC2 Amazon-Benutzerhandbuch.

Über dieses Tutorial

Wenn Sie diesem Tutorial folgen, verwenden Sie den, AWS SDK für .NET um Folgendes zu tun:

- Erstellt eine Spot-Instance-Anforderung
- Ermitteln Sie, wann die Spot-Instance-Anfrage erfüllt wurde
- Stornieren Sie die Spot-Instance-Anfrage
- Beenden von dazugehörigen Instances

Die folgenden Abschnitte enthalten Auszüge und andere Informationen für dieses Beispiel. Der vollständige Code für das Beispiel wird hinter den Codefragmenten angezeigt und kann unverändert erstellt und ausgeführt werden.

Themen

- Voraussetzungen
- Sammeln Sie, was Sie benötigen
- Eine Spot-Instance-Anfrage erstellen
- Ermitteln Sie den Status Ihrer Spot-Instance-Anfrage
- Bereinigen Sie Ihre Spot-Instance-Anfragen
- Bereinigen Sie Ihre Spot-Instances
- Vollständiger Code
- Weitere Überlegungen

Voraussetzungen

Informationen zu den Voraussetzungen APIs und den Voraussetzungen finden Sie im übergeordneten Abschnitt ()Mit Amazon arbeiten EC2.

Sammeln Sie, was Sie benötigen

Um eine Spot-Instance-Anfrage zu erstellen, benötigen Sie mehrere Dinge.

- Die Anzahl der Instances und ihr Instance-Typ. Es stehen mehrere Instance-Typen zur Auswahl. Weitere Informationen finden Sie unter EC2 Amazon-Instance-Typen im EC2 Amazon-Benutzerhandbuch. Weitere Informationen finden Sie unter Details zum Instanztyp und Instanztyp-Explorer.

  Die Standardzahl für dieses Tutorial ist 1.

- Das Amazon Machine Image (AMI), das zur Erstellung der Instance verwendet wird. Weitere Informationen dazu AMIs finden Sie unter Amazon Machine Images (AMIs) im EC2 Amazon-Benutzerhandbuch. Insbesondere siehe Find an AMI and Shared AMIs.

- Der Höchstpreis, den Sie pro Instance-Stunde zu zahlen bereit sind. Sie können die Preise für alle Instance-Typen (sowohl für On-Demand-Instances als auch für Spot-Instances) auf der EC2 Amazon-Preisseite einsehen. Der Standardpreis für dieses Tutorial wird später erklärt.

- Wenn Sie eine Remoteverbindung zu einer Instance herstellen möchten, eine Sicherheitsgruppe mit der entsprechenden Konfiguration und den entsprechenden Ressourcen. Eine Beschreibung

dazu finden Sie unter Arbeiten mit Sicherheitsgruppen in Amazon EC2 und die Informationen zum Sammeln der benötigten Daten und zum Herstellen einer Verbindung zu einer Instanz finden Sie unterStarten einer EC2 Amazon-Instance. Der Einfachheit halber wird in diesem Tutorial die Sicherheitsgruppe mit dem Namen default verwendet, über die alle neueren AWS Konten verfügen.

Es gibt viele Möglichkeiten zum Anfordern von Spot-Instances. Die folgenden Strategien sind gebräuchlich:

- Stellen Sie Anfragen, die mit Sicherheit weniger kosten als On-Demand-Preise.
- Stellen Sie Anfragen auf der Grundlage des Werts der resultierenden Berechnung.
- Stellen Sie Anfragen, um so schnell wie möglich Rechenkapazität zu erwerben.

Die folgenden Erläuterungen beziehen sich auf die Preisentwicklung für Spot-Instances im EC2 Amazon-Benutzerhandbuch.

Reduzieren Sie die Kosten auf einen Wert unter

Sie haben eine Stapelverarbeitungsaufgabe, die einige Stunden oder Tage laufen wird. Allerdings sind Sie flexibel, was den Start und Abschluss angeht. Sie möchten die Aufgabe nach Möglichkeit günstiger als mit On-Demand-Instances abschließen.

Sie untersuchen den Spot-Price-Verlauf für Instance-Typen, indem Sie entweder die EC2 Amazon-Konsole oder die EC2 Amazon-API verwenden. Nachdem Sie den Preisverlauf für Ihren gewünschten Instance-Typ in einer bestimmten Availability Zone analysiert haben, gibt es zwei alternative Ansätze für Ihr Gebot:

- Geben Sie eine Anfrage am oberen Ende der Spanne der Spot-Preise an, die immer noch unter dem On-Demand-Preis liegen. Gehen Sie davon aus, dass Ihre einmalige Spot-Instance-Anfrage höchstwahrscheinlich erfüllt wird und für genügend aufeinanderfolgende Rechenzeit ausgeführt wird, um den Job abzuschließen.
- Oder Sie geben ein Gebot am unteren Ende der Preisskala ab und planen die Kombination vieler Instances, die im Laufe der Zeit über eine persistente Anforderung starten. Die Instances würden zusammengenommen lange genug laufen, um die Aufgabe sogar zu noch geringeren Gesamtkosten abzuschließen.

Zahlen Sie nicht mehr als den Wert des Ergebnisses

Sie haben eine Aufgabe zur Datenverarbeitung, die ausgeführt werden soll. Sie kennen den Wert der Ergebnisse des Jobs gut genug, um zu wissen, wie viel sie in Bezug auf die Computerkosten wert sind.

Nachdem Sie die Spot-Price-Historie für Ihren Instance-Typ analysiert haben, wählen Sie einen Preis, bei dem die Kosten für die Rechenzeit nicht höher sind als der Wert der Ergebnisse des Jobs. Sie erstellen eine persistente Anforderung und lassen es zwischenzeitlich laufen, sobald der Spot-Preis Ihr Gebot erreicht ist oder darunter sinkt.

Erwerben Sie schnell Rechenkapazität

Sie haben einen unvorhergesehenen, kurzfristigen Bedarf an zusätzlicher Kapazität, der über On-Demand-Instances nicht verfügbar ist. Nachdem Sie den Spot-Price-Verlauf für Ihren Instance-Typ analysiert haben, wählen Sie einen Preis, der über dem höchsten historischen Preis liegt, um die Wahrscheinlichkeit zu erhöhen, dass Ihre Anfrage schnell bearbeitet wird, und setzen die Datenverarbeitung fort, bis sie abgeschlossen ist.

Nachdem Sie alles Notwendige zusammengetragen und eine Strategie ausgewählt haben, können Sie eine Spot-Instance anfordern. Für dieses Tutorial wird der standardmäßige Höchstpreis einer Spot-Instance mit dem On-Demand-Preis (in diesem Tutorial 0,003 USD) festgelegt. Indem der Preises auf diese Weise festgelegt wird, erhöhen sich die Chancen zur Erfüllung der Anfrage.

Eine Spot-Instance-Anfrage erstellen

Der folgende Ausschnitt zeigt Ihnen, wie Sie eine Spot-Instance-Anfrage mit den zuvor gesammelten Elementen erstellen.

Das Beispiel am Ende dieses Themas zeigt, wie dieses Snippet verwendet wird.

```
//
// 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];
  }
```

Der wichtige Wert, der von dieser Methode zurückgegeben wird, ist die Spot-Instance-Anforderungs-ID, die im `SpotInstanceRequestId` Element des zurückgegebenen SpotInstanceRequestObjekts enthalten ist.

> **Note**
>
> Ihnen werden alle Spot-Instances in Rechnung gestellt, die gestartet werden. Um unnötige Kosten zu vermeiden, sollten Sie alle Anfragen stornieren und alle Instances beenden.

Ermitteln Sie den Status Ihrer Spot-Instance-Anfrage

Der folgende Ausschnitt zeigt Ihnen, wie Sie Informationen zu Ihrer Spot-Instance-Anfrage erhalten. Sie können diese Informationen verwenden, um bestimmte Entscheidungen in Ihrem Code zu treffen, z. B. ob Sie weiterhin auf die Erfüllung einer Spot-Instance-Anfrage warten möchten.

Das Beispiel am Ende dieses Themas zeigt, wie dieser Codeausschnitt verwendet wird.

```
  //
  // 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];
  }
```

Die Methode gibt Informationen über die Spot-Instance-Anfrage zurück, z. B. die Instance-ID, ihren Status und den Statuscode. Weitere Informationen zu den Statuscodes für Spot-Instance-Anfragen finden Sie unter Spot-Anforderungsstatus im EC2 Amazon-Benutzerhandbuch.

Bereinigen Sie Ihre Spot-Instance-Anfragen

Wenn Sie Spot-Instances nicht mehr anfordern müssen, ist es wichtig, alle ausstehenden Anfragen zu stornieren, um zu verhindern, dass diese Anfragen erneut bearbeitet werden. Der folgende Ausschnitt zeigt Ihnen, wie Sie eine Spot-Instance-Anfrage stornieren.

Das Beispiel am Ende dieses Themas zeigt, wie dieses Snippet verwendet wird.

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

Bereinigen Sie Ihre Spot-Instances

Um unnötige Kosten zu vermeiden, ist es wichtig, dass Sie alle Instances beenden, die über Spot-Instance-Anfragen gestartet wurden. Durch das einfache Stornieren von Spot-Instance-Anfragen werden Ihre Instances nicht beendet, was bedeutet, dass Ihnen diese weiterhin in Rechnung gestellt werden. Der folgende Ausschnitt zeigt Ihnen, wie Sie eine Instance beenden, nachdem Sie die Instance-ID für eine aktive Spot-Instance erhalten haben.

Das Beispiel am Ende dieses Themas zeigt, wie dieses Snippet verwendet wird.

```
  //
  // 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");
        }
      }
    }
```

Vollständiger Code

Im folgenden Codebeispiel werden die zuvor beschriebenen Methoden aufgerufen, um eine Spot-Instance-Anfrage zu erstellen und abzubrechen und eine Spot-Instance zu beenden.

SDK-Referenzen

NuGet Pakete:

- AWSSDK.EC2

Elemente der Programmierung:

- Namespace Amazon. EC2

  Klasse EC2Amazon-Kunde

  Klasse InstanceType

- Namespace Amazon. EC2.Modell

  Klasse CancelSpotInstanceRequestsRequest

  Klasse DescribeSpotInstanceRequestsRequest

  Klasse DescribeSpotInstanceRequestsResponse

  Klasse InstanceStateChange

  Klasse LaunchSpecification

  Klasse RequestSpotInstancesRequest

  Klasse RequestSpotInstancesResponse

  Klasse SpotInstanceRequest

  Klasse TerminateInstancesRequest

  Klasse TerminateInstancesResponse

Der Code

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

Weitere Überlegungen

- Nachdem Sie das Tutorial ausgeführt haben, empfiehlt es sich, sich bei der EC2 Amazon-Konsole anzumelden, um zu überprüfen, ob die Spot-Instance-Anfrage storniert und die Spot-Instance beendet wurde.

Zugriff auf AWS Identity and Access Management (IAM) mit dem AWS SDK für .NET

The AWS SDK für .NET AWS Identity and Access ManagementSupports, ein Webdienst, mit dem AWS Kunden Benutzer und Benutzerberechtigungen verwalten können AWS.

Ein AWS Identity and Access Management (IAM-) Benutzer ist eine Entität, in AWS der Sie eine Entität erstellen. Die Entität stellt eine Person oder Anwendung dar, mit der interagiert. AWS Weitere Informationen zu IAM-Benutzern finden Sie unter IAM-Benutzer und IAM - und STS-Grenzwerte im IAM-Benutzerhandbuch.

Sie gewähren einem Benutzer Berechtigungen, indem Sie eine IAM-Richtlinie erstellen. Die Richtlinie enthält ein Richtliniendokument, in dem die Aktionen aufgeführt sind, die ein Benutzer ausführen kann, und die Ressourcen, auf die sich diese Aktionen auswirken können. Weitere Informationen zu IAM-Richtlinien finden Sie unter Richtlinien und Berechtigungen im IAM-Benutzerhandbuch.

> **Warning**
>
> Um Sicherheitsrisiken zu vermeiden, sollten Sie IAM-Benutzer nicht zur Authentifizierung verwenden, wenn Sie speziell entwickelte Software entwickeln oder mit echten Daten arbeiten. Verwenden Sie stattdessen den Verbund mit einem Identitätsanbieter wie AWS IAM Identity Center.

APIs

Das AWS SDK für .NET bietet APIs IAM-Clients. Sie APIs ermöglichen es Ihnen, mit IAM-Funktionen wie Benutzern, Rollen und Zugriffsschlüsseln zu arbeiten.

Dieser Abschnitt enthält eine kleine Anzahl von Beispielen, die Ihnen zeigen, welchen Mustern Sie bei der Arbeit mit diesen APIs Modellen folgen können. Den vollständigen Satz von APIs finden Sie in der AWS SDK für .NET API-Referenz (und scrollen Sie zu „Amazon. IdentityManagement„).

Dieser Abschnitt enthält auch ein Beispiel, das Ihnen zeigt, wie Sie EC2 Amazon-Instances eine IAM-Rolle zuordnen, um die Verwaltung von Anmeldeinformationen zu vereinfachen.

Die IAM APIs werden bereitgestellt von. AWSSDK IdentityManagement NuGetPaket.

Voraussetzungen

Bevor Sie beginnen, stellen Sie sicher, dass Sie Ihre Umgebung eingerichtet und Ihr Projekt konfiguriert haben. Lesen Sie auch die Informationen unterVerwenden der SDK.

Themen

Themen

- Von IAM verwaltete Richtlinien aus JSON erstellen
- Zeigen Sie das Richtliniendokument einer von IAM verwalteten Richtlinie an
- Zugriff mithilfe einer IAM-Rolle gewähren

Von IAM verwaltete Richtlinien aus JSON erstellen

Dieses Beispiel zeigt Ihnen, wie Sie mithilfe von eine IAM-verwaltete Richtlinie aus einem bestimmten Richtliniendokument in JSON erstellen können. AWS SDK für .NET Die Anwendung erstellt ein IAM-Client-Objekt, liest das Richtliniendokument aus einer Datei und erstellt dann die Richtlinie.

> **Note**
>
> Ein Beispiel für ein Richtliniendokument in JSON finden Sie in den zusätzlichen Überlegungen am Ende dieses Themas.

Die folgenden Abschnitte enthalten Auszüge aus diesem Beispiel. Der vollständige Code für das Beispiel wird danach angezeigt und kann unverändert erstellt und ausgeführt werden.

Themen

- Erstellen der -Richtlinie
- Vollständiger Code
- Weitere Überlegungen

Erstellen der -Richtlinie

Das folgende Snippet erstellt eine von IAM verwaltete Richtlinie mit dem angegebenen Namen und dem Richtliniendokument.

Das Beispiel am Ende dieses Themas zeigt, wie dieser Codeausschnitt verwendet wird.

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

Vollständiger Code

Dieser Abschnitt enthält relevante Referenzen und den vollständigen Code für dieses Beispiel.

SDK-Referenzen

NuGet Pakete:

- AWSSDK.IdentityManagement

Elemente der Programmierung:

- Namespace Amazon. IdentityManagement

  Klasse AmazonIdentityManagementServiceClient
- Namespace Amazon. IdentityManagement.Modell

  Klasse CreatePolicyRequest

  Klasse CreatePolicyResponse

Der Code

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Threading.Tasks;
using Amazon.IdentityManagement;
using Amazon.IdentityManagement.Model;

namespace IamCreatePolicyFromJson
{
  // = = =
```

```
      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.");
    }
  }
```

```
 // = = =
```

```
          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);
    }
  }

}
```

Weitere Überlegungen

- Im Folgenden finden Sie ein Beispiel für ein Richtliniendokument, das Sie in eine JSON-Datei kopieren und als Eingabe für diese Anwendung verwenden können:

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": "*"
    }
  ]
}
```

- Sie können in der IAM-Konsole überprüfen, ob die Richtlinie erstellt wurde. Wählen Sie in der Dropdownliste Filterrichtlinien die Option Vom Kunden verwaltet aus. Löschen Sie die Richtlinie, wenn Sie sie nicht mehr benötigen.

- Weitere Informationen zur Erstellung von Richtlinien finden Sie unter Erstellen von IAM-Richtlinien und in der IAM-JSON-Richtlinienreferenz im IAM-Benutzerhandbuch

Zeigen Sie das Richtliniendokument einer von IAM verwalteten Richtlinie an

Dieses Beispiel zeigt Ihnen, wie Sie mit AWS SDK für .NET dem ein Richtliniendokument anzeigen können. Die Anwendung erstellt ein IAM-Client-Objekt, sucht die Standardversion der angegebenen IAM-verwalteten Richtlinie und zeigt dann das Richtliniendokument in JSON an.

Die folgenden Abschnitte enthalten Auszüge aus diesem Beispiel. Der vollständige Code für das Beispiel wird danach angezeigt und kann unverändert erstellt und ausgeführt werden.

Themen

- Finden Sie die Standardversion
- Zeigen Sie das Richtliniendokument an
- Vollständiger Code

Finden Sie die Standardversion

Das folgende Snippet findet die Standardversion der angegebenen IAM-Richtlinie.

Das Beispiel am Ende dieses Themas zeigt, wie dieses Snippet verwendet wird.

```
//
// 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;
}
```

Zeigen Sie das Richtliniendokument an

Der folgende Ausschnitt zeigt das Richtliniendokument der angegebenen IAM-Richtlinie in JSON an.

Das Beispiel am Ende dieses Themas zeigt, wie dieses Snippet verwendet wird.

```
    //
    // 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)}");
    }
```

Vollständiger Code

Dieser Abschnitt enthält relevante Referenzen und den vollständigen Code für dieses Beispiel.

SDK-Referenzen

NuGet Pakete:

- AWSSDK.IdentityManagement

Elemente der Programmierung:

- Namespace Amazon. IdentityManagement

  Klasse AmazonIdentityManagementServiceClient
- Namespace Amazon. IdentityManagement.Modell

  Klasse GetPolicyVersionRequest

Klasse GetPolicyVersionResponse

Klasse ListPolicyVersionsRequest

Klasse ListPolicyVersionsResponse

Klasse PolicyVersion

Der Code

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

Zugriff mithilfe einer IAM-Rolle gewähren

Dieses Tutorial zeigt Ihnen, wie Sie die verwenden, AWS SDK für .NET um IAM-Rollen auf EC2 Amazon-Instances zu aktivieren.

Übersicht

Alle Anfragen an AWS müssen kryptografisch signiert werden, wobei die Anmeldeinformationen verwendet werden müssen, die von ausgestellt wurden. AWS Daher benötigen Sie eine Strategie zur Verwaltung von Anmeldeinformationen für Anwendungen, die auf EC2 Amazon-Instances ausgeführt werden. Sie müssen diese Anmeldeinformationen sicher verteilen, speichern und rotieren, aber auch dafür sorgen, dass sie für die Anwendungen zugänglich sind.

Mit IAM-Rollen können Sie diese Anmeldeinformationen effektiv verwalten. Sie erstellen eine IAM-Rolle und konfigurieren sie mit den Berechtigungen, die eine Anwendung benötigt, und fügen diese Rolle dann einer EC2 Instanz hinzu. Weitere Informationen zu den Vorteilen der Verwendung von IAM-Rollen finden Sie unter IAM-Rollen für Amazon EC2 im _EC2 Amazon-Benutzerhandbuch. Weitere Informationen zu IAM-Rollen finden Sie auch im IAM-Benutzerhandbuch.

Bei einer Anwendung, die mit dem erstellt wurde AWS SDK für .NET, sucht das Objekt, wenn die Anwendung ein Client-Objekt für einen AWS Dienst erstellt, nach Anmeldeinformationen aus verschiedenen möglichen Quellen. Die Reihenfolge, in der gesucht wird, wird in Auflösung von Anmeldeinformationen und Profilen angezeigt.

Wenn das Client-Objekt keine Anmeldeinformationen aus einer anderen Quelle findet, ruft es temporäre Anmeldeinformationen ab, die dieselben Berechtigungen haben wie diejenigen, die für die IAM-Rolle konfiguriert wurden und sich in den Metadaten der EC2 Instanz befinden. Diese Anmeldeinformationen werden verwendet, um vom Client-Objekt AWS aus Aufrufe zu tätigen.

Über dieses Tutorial

Wenn Sie diesem Tutorial folgen, verwenden Sie die AWS SDK für .NET (und andere Tools), um eine EC2 Amazon-Instance mit einer angehängten IAM-Rolle zu starten, und sehen dann eine Anwendung auf der Instance, die die Berechtigungen der IAM-Rolle verwendet.

Themen

- Erstellen Sie eine Amazon S3 S3-Beispielanwendung
- Erstellen einer IAM-Rolle
- Starten Sie eine EC2 Instance und fügen Sie die IAM-Rolle hinzu
- Connect zur EC2 Instanz her
- Führen Sie die Beispielanwendung auf der EC2 Instance aus
- Bereinigen

Erstellen Sie eine Amazon S3 S3-Beispielanwendung

Diese Beispielanwendung ruft ein Objekt von Amazon S3 ab. Um die Anwendung auszuführen, benötigen Sie Folgendes:

- Ein Amazon S3 S3-Bucket, der eine Textdatei enthält.
- AWS Anmeldeinformationen auf Ihrem Entwicklungscomputer, die Ihnen den Zugriff auf den Bucket ermöglichen.

Informationen zum Erstellen eines Amazon S3 S3-Buckets und zum Hochladen eines Objekts finden Sie im Amazon Simple Storage Service-Benutzerhandbuch. Informationen zu AWS Anmeldeinformationen finden Sie unterAuthentifizieren des mit AWS SDK für .NETAWS.

Erstellen Sie ein .NET Core-Projekt mit dem folgenden Code. Testen Sie dann die Anwendung auf Ihrem Entwicklungscomputer.

> **Note**
>
> Auf Ihrem Entwicklungscomputer ist die.NET Core Runtime installiert, sodass Sie die Anwendung ausführen können, ohne sie zu veröffentlichen. Wenn Sie später in diesem Tutorial eine EC2 Instanz erstellen, können Sie sich dafür entscheiden, die .NET Core Runtime auf der Instanz zu installieren. Dies bietet Ihnen eine ähnliche Erfahrung und eine geringere Dateiübertragung.

Sie können sich jedoch auch dafür entscheiden, die.NET Core Runtime nicht auf der Instanz zu installieren. Wenn Sie sich für diese Vorgehensweise entscheiden, müssen Sie die Anwendung veröffentlichen, sodass bei der Übertragung auf die Instanz alle Abhängigkeiten berücksichtigt werden.

SDK-Referenzen

NuGet Pakete:

- AWSSDKS3

Elemente der Programmierung:

- Namespace Amazon.S3

  Klasse AmazonS3Client
- Namespace Amazon.S3.Model

  Klasse GetObjectResponse

Der Code

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Threading.Tasks;
using Amazon.S3;
using Amazon.S3.Model;

namespace S3GetTextItem
{
  // = = =
```

```
      if(parsedArgs.Count == 0)
      {
        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.");
    }
  }


  // = = =
```

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

```
}
```

Wenn Sie möchten, können Sie die Anmeldeinformationen, die Sie auf Ihrem Entwicklungscomputer verwenden, vorübergehend entfernen, um zu sehen, wie die Anwendung reagiert. (Stellen Sie jedoch sicher, dass Sie die Anmeldeinformationen wiederherstellen, wenn Sie fertig sind.)

Erstellen einer IAM-Rolle

Erstellen Sie eine IAM-Rolle, die über die entsprechenden Berechtigungen für den Zugriff auf Amazon S3 verfügt.

1. Öffnen Sie die IAM-Konsole.
2. Wählen Sie im Navigationsbereich Rollen und dann Rolle erstellen aus.
3. Wählen Sie AWS Dienst aus, suchen und wählen EC2Sie dann Weiter: Berechtigungen aus.
4. Suchen Sie unter Richtlinien zum Anhängen von Berechtigungen nach AmazonS3 und wählen Sie es aus. ReadOnlyAccess Überprüfen Sie die Richtlinie, falls Sie möchten, und wählen Sie dann Weiter: Tags aus.
5. Fügen Sie Schlagworte hinzu, wenn Sie möchten, und wählen Sie dann Weiter: Überprüfen.
6. Geben Sie einen Namen und eine Beschreibung für die Rolle ein und wählen Sie dann Create role aus. Merken Sie sich diesen Namen, da Sie ihn benötigen, wenn Sie Ihre EC2 Instance starten.

Starten Sie eine EC2 Instance und fügen Sie die IAM-Rolle hinzu

Starten Sie eine EC2 Instance mit der IAM-Rolle, die Sie zuvor erstellt haben. Sie können dies auf folgende Weise tun.

- Verwenden der EC2 Konsole

  Informationen zum Starten einer Instance mithilfe der EC2 Konsole finden Sie unter Starten einer Instance mithilfe des Assistenten zum Starten einer Instance im EC2 Amazon-Benutzerhandbuch.

  Wenn Sie sich die Startseite ansehen, sollten Sie zumindest den Bereich mit den erweiterten Details erweitern, sodass Sie die IAM-Rolle angeben können, die Sie zuvor im IAM-Instance-Profil erstellt haben.

- Verwenden Sie den AWS SDK für .NET

Informationen dazu finden Sie unterStarten einer EC2 Amazon-Instance, auch Weitere Überlegungen am Ende dieses Themas.

Um eine EC2 Instance zu starten, der eine IAM-Rolle zugewiesen ist, muss die Konfiguration eines IAM-Benutzers bestimmte Berechtigungen enthalten. Weitere Informationen zu den erforderlichen Berechtigungen finden Sie im EC2 Amazon-Benutzerhandbuch unter Erteilen einer Benutzerberechtigung zur Übergabe einer IAM-Rolle an eine Instance.

Connect zur EC2 Instanz her

Stellen Sie eine Verbindung mit der EC2 Instanz her, sodass Sie die Beispielanwendung auf sie übertragen und die Anwendung dann ausführen können. Sie benötigen die Datei, die den privaten Teil des key pair enthält, das Sie zum Starten der Instance verwendet haben, d. h. die PEM-Datei.

Informationen zum Herstellen einer Verbindung mit einer Instance finden Sie unter Connect to your Linux Instance oder Connect to your Windows Instance im EC2 Amazon-Benutzerhandbuch. Wenn Sie eine Verbindung herstellen, tun Sie dies so, dass Sie Dateien von Ihrem Entwicklungscomputer auf Ihre Instance übertragen können.

Wenn Sie Visual Studio unter Windows verwenden, können Sie auch mithilfe des Toolkit for Visual Studio eine Verbindung mit der Instanz herstellen. Weitere Informationen finden Sie unter Verbindung zu einer EC2 Amazon-Instance herstellen im AWS Toolkit for Visual Studio Benutzerhandbuch.

Führen Sie die Beispielanwendung auf der EC2 Instance aus

1. Kopieren Sie die Anwendungsdateien von Ihrem lokalen Laufwerk auf Ihre Instanz.

   Welche Dateien Sie übertragen, hängt davon ab, wie Sie die Anwendung erstellt haben und ob auf Ihrer Instanz die.NET Core Runtime installiert ist. Informationen zum Übertragen von Dateien auf Ihre Instance finden Sie unter Connect zu Ihrer Linux-Instance herstellen (siehe den entsprechenden Unterabschnitt) oder Dateien auf Windows-Instances übertragen im EC2 Amazon-Benutzerhandbuch.
2. Starten Sie die Anwendung und stellen Sie sicher, dass sie mit den gleichen Ergebnissen wie auf Ihrem Entwicklungscomputer ausgeführt wird.
3. Stellen Sie sicher, dass die Anwendung die von der IAM-Rolle bereitgestellten Anmeldeinformationen verwendet.

   a. Öffnen Sie die EC2 Amazon-Konsole.

b. Wählen Sie die Instance aus und trennen Sie die IAM-Rolle über Aktionen, Instance-Einstellungen, IAM-Rolle anhängen/ersetzen.

c. Führen Sie die Anwendung erneut aus und stellen Sie fest, dass sie einen Autorisierungsfehler zurückgibt.

Bereinigen

Wenn Sie mit diesem Tutorial fertig sind und die von Ihnen erstellte EC2 Instanz nicht mehr benötigen, sollten Sie die Instanz unbedingt beenden, um unerwünschte Kosten zu vermeiden. Sie können dies in der EC2 Amazon-Konsole oder programmgesteuert tun, wie unter beschrieben. Beenden einer Amazon-Instance EC2 Wenn Sie möchten, können Sie auch andere Ressourcen löschen, die Sie für dieses Tutorial erstellt haben. Dazu können eine IAM-Rolle, ein EC2 Schlüsselpaar und eine PEM-Datei, eine Sicherheitsgruppe usw. gehören.

Verwenden des Internetspeichers von Amazon Simple Storage Service

Das AWS SDK für .NET unterstützt Amazon S3, einen Speicher für das Internet. Der Service ist darauf ausgelegt, Cloud Computing für Entwickler zu erleichtern.

APIs

Das AWS SDK für .NET bietet APIs Amazon S3 S3-Clients. Sie APIs ermöglichen es Ihnen, mit Amazon S3 S3-Ressourcen wie Buckets und Artikeln zu arbeiten. Den vollständigen Satz von APIs für Amazon S3 finden Sie im Folgenden:

- AWS SDK für .NET API-Referenz (und scrollen Sie zu „Amazon.S3").
- Amazon.Extensions.S3.Dokumentation zur Verschlüsselung

Die Amazon S3 APIs werden in den folgenden NuGet Paketen bereitgestellt:

- AWSSDKS3
- Amazon.Extensions.S3. Verschlüsselung

Voraussetzungen

Bevor Sie beginnen, stellen Sie sicher, dass Sie Ihre Umgebung eingerichtet und Ihr Projekt konfiguriert haben. Lesen Sie auch die Informationen unterVerwenden der SDK.

Beispiele in diesem Dokument

Die folgenden Themen in diesem Dokument zeigen Ihnen, wie Sie den verwenden AWS SDK für .NET , um mit Amazon S3 zu arbeiten.

- Verwendung von KMS-Schlüsseln für die S3-Verschlüsselung

Beispiele in anderen Dokumenten

Die folgenden Links zum Amazon S3-Entwicklerhandbuch enthalten weitere Beispiele für die Verwendung von für AWS SDK für .NET die Arbeit mit Amazon S3.

> **Note**
>
> Diese Beispiele und zusätzliche Überlegungen zur Programmierung wurden zwar für Version 3 AWS SDK für .NET des.NET-Frameworks entwickelt, sie eignen sich aber auch für spätere Versionen AWS SDK für .NET des.NET Core. Manchmal sind kleine Anpassungen im Code erforderlich.

Amazon S3 S3-Programmierbeispiele

- Verwaltung ACLs
- Erstellen eines Buckets
- Hochladen eines Objekts
- Mehrteiliger Upload mit der High-Level-API (Amazon.S3.Transfer. TransferUtility)
- Mehrteilige Uploads mit der Low-Level-API
- Auflisten von Objekten
- Auflisten von Schlüsseln
- Abrufen eines Objekts
- Kopieren eines Objekts
- Kopieren eines Objekts unter Verwendung der API für mehrteilige Uploads
- Löschen eines Objekts
- Löschen von mehreren Objekten

- Wiederherstellen eines Objekts
- Konfigurieren eines Buckets für Benachrichtigungen
- Verwalten des Lebenszyklus eines Objekts
- Generieren einer vorsignierten Objekt-URL
- Verwalten von Websites
- Cross-Origin Resource Sharing (CORS) aktivieren

Zusätzliche Überlegungen zur Programmierung

- Verwenden der AWS SDK für .NET für Amazon S3 S3-Programmierung
- Erstellen von Anfragen unter Verwendung temporärer Anmeldeinformationen für IAM-Benutzer
- Anfragen unter Verwendung temporärer Anmeldeinformationen verbundener Benutzer stellen
- Festlegen einer serverseitigen Verschlüsselung
- Festlegen einer serverseitigen Verschlüsselung mit vom Kunden bereitgestellten Verschlüsselungsschlüsseln

Verwendung von AWS KMS Schlüsseln für die Amazon S3 S3-Verschlüsselung in der AWS SDK für .NET

Dieses Beispiel zeigt Ihnen, wie Sie AWS Key Management Service Schlüssel verwenden, um Amazon S3 S3-Objekte zu verschlüsseln. Die Anwendung erstellt einen Kundenhauptschlüssel (CMK) und verwendet ihn, um ein AmazonS3 EncryptionClient V2-Objekt für die clientseitige Verschlüsselung zu erstellen. Die Anwendung verwendet diesen Client, um ein verschlüsseltes Objekt aus einer bestimmten Textdatei in einem vorhandenen Amazon S3 S3-Bucket zu erstellen. Anschließend entschlüsselt sie das Objekt und zeigt seinen Inhalt an.

> **Warning**
>
> Eine ähnliche Klasse namens `AmazonS3EncryptionClient` ist veraltet und weniger sicher als die Klasse. `AmazonS3EncryptionClientV2` Informationen zur Migration von vorhandenem Code, der verwendet`AmazonS3EncryptionClient`, finden Sie unter. Migration des S3-Verschlüsselungsclients

Themen

- Verschlüsselungsmaterial erstellen
- Ein Amazon S3 S3-Objekt erstellen und verschlüsseln
- Vollständiger Code
- Weitere Überlegungen

Verschlüsselungsmaterial erstellen

Mit dem folgenden Codeausschnitt wird ein `EncryptionMaterials` Objekt erstellt, das eine KMS-Schlüssel-ID enthält.

Das Beispiel am Ende dieses Themas zeigt, wie dieser Codeausschnitt verwendet wird.

```
     // 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);
```

Ein Amazon S3 S3-Objekt erstellen und verschlüsseln

Der folgende Ausschnitt erstellt ein `AmazonS3EncryptionClientV2` Objekt, das die zuvor erstellten Verschlüsselungsmaterialien verwendet. Anschließend verwendet es den Client, um ein neues Amazon S3 S3-Objekt zu erstellen und zu verschlüsseln.

Das Beispiel am Ende dieses Themas zeigt, wie dieses Snippet verwendet wird.

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

Vollständiger Code

Dieser Abschnitt enthält relevante Referenzen und den vollständigen Code für dieses Beispiel.

SDK-Referenzen

NuGet Pakete:

- Amazon.Extensions.S3. Verschlüsselung

Elemente der Programmierung:

- Namespace Amazon.Extensions.S3.Verschlüsselung

  Klasse AmazonS3 V2 EncryptionClient

  Klasse AmazonS3 V2 CryptoConfiguration

  Klasse CryptoStorageMode

  Klasse EncryptionMaterialsV2
- Namespace Amazon.Extensions.S3.Encryption.Primitives

  Klasse KmsType

- Namespace Amazon.S3.Model

  Klasse GetObjectRequest

  Klasse GetObjectResponse

  Klasse PutObjectRequest
- Namespace Amazon. KeyManagementService

  Klasse AmazonKeyManagementServiceClient
- Namespace Amazon. KeyManagementService.Modell

  Klasse CreateKeyRequest

  Klasse CreateKeyResponse

Der Code

```
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
{
  // = = =
```

```
      {
        PrintHelp();
        return;
      }

      // Get the application arguments from the parsed list
      string bucketName =
        CommandLine.GetArgument(parsedArgs, null, "-b", "--bucket-name");
      string fileName =
        CommandLine.GetArgument(parsedArgs, null, "-f", "--file-name");
      string itemName =
        CommandLine.GetArgument(parsedArgs, null, "-i", "--item-name");
      if(string.IsNullOrEmpty(bucketName) || (string.IsNullOrEmpty(fileName)))
        CommandLine.ErrorExit(
          "\nOne or more of the required arguments is missing or incorrect." +
          "\nRun the command with no arguments to see help.");
      if(!File.Exists(fileName))
        CommandLine.ErrorExit($"\nThe given file {fileName} doesn't exist.");
      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.");
    }

  }

  // = = =
```

```
 // (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);
    }
  }

}
```

Weitere Überlegungen

- Sie können die Ergebnisse dieses Beispiels überprüfen. Rufen Sie dazu die Amazon S3 S3-Konsole auf und öffnen Sie den Bucket, den Sie der Anwendung zur Verfügung gestellt haben. Suchen Sie dann das neue Objekt, laden Sie es herunter und öffnen Sie es in einem Texteditor.

- Die AmazonS3 EncryptionClient V2-Klasse implementiert dieselbe Schnittstelle wie die `AmazonS3Client` Standardklasse. Dies erleichtert die Portierung Ihres Codes in die `AmazonS3EncryptionClientV2` Klasse, sodass die Verschlüsselung und Entschlüsselung automatisch und transparent im Client erfolgt.

- Ein Vorteil der Verwendung eines AWS KMS Schlüssels als Hauptschlüssel besteht darin, dass Sie Ihre eigenen Hauptschlüssel nicht speichern und verwalten müssen. Dies erfolgt durch. AWS Ein zweiter Vorteil besteht darin, dass die `AmazonS3EncryptionClientV2` Klasse von mit der `AmazonS3EncryptionClientV2` Klasse von interoperabel AWS SDK für .NET ist. AWS SDK für Java Das heißt, Sie können mit dem verschlüsseln AWS SDK für Java und mit dem entschlüsseln und AWS SDK für .NET umgekehrt.

  > **Note**
  >
  > Die `AmazonS3EncryptionClientV2` Klasse von AWS SDK für .NET unterstützt KMS-Masterschlüssel nur, wenn sie im Metadatenmodus ausgeführt wird. Der Befehlsdateimodus der `AmazonS3EncryptionClientV2` Klasse von AWS SDK für .NET ist nicht kompatibel mit der `AmazonS3EncryptionClientV2` Klasse von AWS SDK für Java.

- Weitere Informationen zur clientseitigen Verschlüsselung mit der `AmazonS3EncryptionClientV2` Klasse und zur Funktionsweise der Envelope-Verschlüsselung finden Sie unter Clientseitige Datenverschlüsselung mit AWS SDK für .NET und Amazon S3.

Senden von Benachrichtigungen aus der Cloud mit Amazon Simple Notification Service

> **Note**
>
> Die Informationen in diesem Thema beziehen sich speziell auf Projekte, die auf .NET Framework und AWS SDK für .NET Version 3.3 und früher basieren.

Der AWS SDK für .NET unterstützt Amazon Simple Notification Service (Amazon SNS), einen Webservice, der es Anwendungen, Endbenutzern und Geräten ermöglicht, sofort Benachrichtigungen aus der Cloud zu senden. Weitere Informationen finden Sie unter Amazon SNS.

Auflisten Ihrer Amazon SNS SNS-Themen

Das folgende Beispiel zeigt, wie Sie Ihre Amazon SNS SNS-Themen, die Abonnements für jedes Thema und die Attribute für jedes Thema auflisten. In diesem Beispiel wird die Standardeinstellung AmazonSimpleNotificationServiceClientverwendet.

```
// 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));
```

Eine Nachricht an ein Amazon SNS SNS-Thema senden

Das folgende Beispiel zeigt, wie eine Nachricht an ein Amazon SNS SNS-Thema gesendet wird. Das Beispiel verwendet ein Argument, den ARN des Amazon SNS SNS-Themas.

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

Das vollständige Beispiel, einschließlich Informationen zum Erstellen und Ausführen des Beispiels über die Befehlszeile, finden Sie unter GitHub.

Senden einer SMS-Nachricht an eine Telefonnummer

Das folgende Beispiel zeigt, wie Sie eine SMS-Nachricht an eine Telefonnummer senden. Das Beispiel verwendet ein Argument, die Telefonnummer, die in einem der beiden in den Kommentaren beschriebenen Formate vorliegen muss.

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

Das vollständige Beispiel, einschließlich Informationen zum Erstellen und Ausführen des Beispiels über die Befehlszeile, finden Sie unter GitHub.

Nachrichtenübermittlung mit Amazon SQS

Der AWS SDK für .NET unterstützt Amazon Simple Queue Service (Amazon SQS), einen Nachrichtenwarteschlangen-Service, der Nachrichten oder Workflows zwischen Komponenten in einem System verarbeitet.

Amazon SQS SQS-Warteschlangen bieten einen Mechanismus, mit dem Sie Nachrichten zwischen Softwarekomponenten wie Microservices, verteilten Systemen und serverlosen Anwendungen senden, speichern und empfangen können. Auf diese Weise können Sie solche Komponenten entkoppeln und müssen kein eigenes Messaging-System entwerfen und betreiben. Informationen zur Funktionsweise von Warteschlangen und Nachrichten in Amazon SQS finden Sie in den Amazon SQS-Tutorials und unter Basic Amazon SQS architecture im Amazon Simple Queue Service Developer Guide.

> **Important**
>
> Aufgrund des verteilten Charakters von Warteschlangen kann Amazon SQS nicht garantieren, dass Sie Nachrichten in der genauen Reihenfolge erhalten, in der sie gesendet wurden. Wenn Sie die Nachrichtenreihenfolge beibehalten müssen, verwenden Sie eine Amazon SQS FIFO-Warteschlange.

APIs

Das AWS SDK für .NET bietet APIs Amazon SQS-Kunden. Sie APIs ermöglichen es Ihnen, mit Amazon SQS SQS-Funktionen wie Warteschlangen und Nachrichten zu arbeiten. Dieser Abschnitt enthält eine kleine Anzahl von Beispielen, die Ihnen zeigen, welchen Mustern Sie bei der Arbeit mit diesen Modellen folgen können. APIs Den vollständigen Satz von finden Sie in der APIs AWS SDK für .NET API-Referenz (und scrollen Sie zu „Amazon.sqs“).

Die Amazon SQS APIs werden durch das AWSSDK NuGet .SQS-Paket bereitgestellt.

Voraussetzungen

Bevor Sie beginnen, stellen Sie sicher, dass Sie Ihre Umgebung eingerichtet und Ihr Projekt konfiguriert haben. Lesen Sie auch die Informationen unterVerwenden der SDK.

Themen

Themen

- Amazon SQS SQS-Warteschlangen erstellen
- Amazon SQS SQS-Warteschlangen aktualisieren
- Löschen von Amazon SQS SQS-Warteschlangen
- Amazon SQS SQS-Nachrichten senden
- Empfangen Amazon SQS SQS-Nachrichten

Amazon SQS SQS-Warteschlangen erstellen

Dieses Beispiel zeigt Ihnen, wie Sie mit dem AWS SDK für .NET eine Amazon SQS SQS-Warteschlange erstellen. Die Anwendung erstellt eine Warteschlange für unzustellbare Briefe, wenn Sie den ARN für eine nicht angeben. Anschließend erstellt sie eine Standard-Nachrichtenwarteschlange, die eine Warteschlange für unzustellbare Nachrichten enthält (die, die Sie angegeben haben oder die, die erstellt wurde).

Wenn Sie keine Befehlszeilenargumente angeben, zeigt die Anwendung einfach Informationen über alle vorhandenen Warteschlangen an.

Die folgenden Abschnitte enthalten Auszüge aus diesem Beispiel. Der vollständige Code für das Beispiel wird danach angezeigt und kann unverändert erstellt und ausgeführt werden.

Themen

- Bestehende Warteschlangen anzeigen
- Erstellen Sie die Warteschlange
- Den ARN einer Warteschlange abrufen
- Vollständiger Code
- Weitere Überlegungen

Bestehende Warteschlangen anzeigen

Der folgende Ausschnitt zeigt eine Liste der vorhandenen Warteschlangen in der Region des SQS-Clients und die Attribute jeder Warteschlange.

Das Beispiel am Ende dieses Themas zeigt, wie dieses Snippet verwendet wird.

```
    //
    // 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}");
    }
```

Erstellen Sie die Warteschlange

Das folgende Snippet erstellt eine Warteschlange. Das Snippet beinhaltet die Verwendung einer Warteschlange für unzustellbare Briefe, für Ihre Warteschlangen ist jedoch nicht unbedingt eine Warteschlange erforderlich.

Das Beispiel am Ende dieses Themas zeigt, wie dieser Codeausschnitt verwendet wird.

```
    //
    // 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;
    }
```

Den ARN einer Warteschlange abrufen

Das folgende Snippet ruft den ARN der Warteschlange ab, die durch die angegebene Warteschlangen-URL identifiziert wird.

Das Beispiel am Ende dieses Themas zeigt, wie dieses Snippet verwendet wird.

```
    //
    // 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;
    }
```

Vollständiger Code

Dieser Abschnitt enthält relevante Referenzen und den vollständigen Code für dieses Beispiel.

SDK-Referenzen

NuGet Pakete:

- AWSSDK.SQS

Elemente der Programmierung:

- Namespace Amazon.sqs

  Klasse Amazon SQSClient

  Klasse QueueAttributeName
- Namespace Amazon.SQS.Model

  Klasse CreateQueueRequest

  Klasse CreateQueueResponse

  Klasse GetQueueAttributesResponse

  Klasse ListQueuesResponse

Der Code

```
using System;
using System.Threading.Tasks;
using System.Collections.Generic;
using Amazon.SQS;
using Amazon.SQS.Model;

namespace SQSCreateQueue
{
  // = = =
```

```
      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);
      }
    }

}
```

Weitere Überlegungen

- Ihr Warteschlangenname muss aus alphanumerischen Zeichen, Bindestrichen und Unterstrichen bestehen.

- Bei Warteschlangennamen und Warteschlangen wird zwischen Groß- und Kleinschreibung unterschieden URLs

- Wenn Sie die Warteschlangen-URL benötigen, aber nur den Warteschlangennamen kennen, verwenden Sie eine der `AmazonSQSClient.GetQueueUrlAsync` Methoden.

- Informationen zu den verschiedenen Warteschlangenattributen, die Sie festlegen können, finden Sie CreateQueueRequestin der AWS SDK für .NET API-Referenz oder SetQueueAttributesin der Amazon Simple Queue Service API-Referenz.

- In diesem Beispiel wird eine lange Abfrage für alle Nachrichten in der von Ihnen erstellten Warteschlange spezifiziert. Dies erfolgt mithilfe des `ReceiveMessageWaitTimeSeconds` Attributs.

  Sie können auch lange Abfragen während eines Aufrufs der `ReceiveMessageAsync` Methoden der SQSClientAmazon-Klasse angeben. Weitere Informationen finden Sie unter Empfangen Amazon SQS SQS-Nachrichten.

  Informationen zu kurzen Abfragen im Vergleich zu langen Abfragen finden Sie unter Kurze und lange Abfragen im Amazon Simple Queue Service Developer Guide.

- Eine Warteschlange für unzustellbare Nachrichten kann von anderen (Quell-) Warteschlangen gezielt für Nachrichten verwendet werden, die nicht erfolgreich verarbeitet wurden. Weitere

Informationen finden Sie unter Amazon SQS Dead-Letter-Warteschlangen im Amazon Simple Queue Service Developer Guide.

- Sie können die Liste der Warteschlangen und die Ergebnisse dieses Beispiels auch in der Amazon SQS SQS-Konsole sehen.

Amazon SQS SQS-Warteschlangen aktualisieren

Dieses Beispiel zeigt Ihnen, wie Sie die verwenden, um eine Amazon SQS SQS-Warteschlange AWS SDK für .NET zu aktualisieren. Nach einigen Prüfungen aktualisiert die Anwendung das angegebene Attribut mit dem angegebenen Wert und zeigt dann alle Attribute für die Warteschlange an.

Wenn nur die Warteschlangen-URL in den Befehlszeilenargumenten enthalten ist, zeigt die Anwendung einfach alle Attribute für die Warteschlange an.

Die folgenden Abschnitte enthalten Auszüge aus diesem Beispiel. Der vollständige Code für das Beispiel wird danach angezeigt und kann unverändert erstellt und ausgeführt werden.

Themen

- Queue-Attribute anzeigen
- Überprüfen Sie den Attributnamen
- Warteschlangenattribut aktualisieren
- Vollständiger Code
- Weitere Überlegungen

Queue-Attribute anzeigen

Das folgende Snippet zeigt die Attribute der Warteschlange, die durch die angegebene Warteschlangen-URL identifiziert werden.

Das Beispiel am Ende dieses Themas zeigt, wie dieses Snippet verwendet wird.

```
//
// 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}");
  }
```

Überprüfen Sie den Attributnamen

Der folgende Ausschnitt validiert den Namen des Attributs, das aktualisiert wird.

Das Beispiel am Ende dieses Themas zeigt, wie dieser Codeausschnitt verwendet wird.

```
  //
  // 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;
  }
```

Warteschlangenattribut aktualisieren

Das folgende Snippet aktualisiert ein Attribut der Warteschlange, das durch die angegebene Warteschlangen-URL identifiziert wird.

Das Beispiel am Ende dieses Themas zeigt, wie dieses Snippet verwendet wird.

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

Vollständiger Code

Dieser Abschnitt enthält relevante Referenzen und den vollständigen Code für dieses Beispiel.

SDK-Referenzen

NuGet Pakete:

- AWSSDK.SQS

Elemente der Programmierung:

- Namespace Amazon.sqs

  Klasse Amazon SQSClient

  Klasse QueueAttributeName
- Namespace Amazon.SQS.Model

  Klasse GetQueueAttributesResponse

Der Code

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.SQS;
using Amazon.SQS.Model;

namespace SQSUpdateQueue
{
  // = = =
```

```
      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.");
    }
  }


  // = = =
```

```
        }

        // 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);
    }
  }

}
```

Weitere Überlegungen

- Um das `RedrivePolicy` Attribut zu aktualisieren, müssen Sie je nach Betriebssystem den gesamten Wert in Anführungszeichen setzen und die Anführungszeichen für die key/value Paare maskieren.

  Unter Windows ist der Wert beispielsweise ähnlich wie folgt aufgebaut:

```
"{\"deadLetterTargetArn\":\"DEAD_LETTER-QUEUE-ARN\",\"maxReceiveCount\":\"10\"}"
```

Löschen von Amazon SQS SQS-Warteschlangen

Dieses Beispiel zeigt Ihnen, wie Sie die verwenden, um eine Amazon SQS SQS-Warteschlange AWS SDK für .NET zu löschen. Die Anwendung löscht die Warteschlange, wartet bis zu einer bestimmten Zeit, bis die Warteschlange gelöscht ist, und zeigt dann eine Liste der verbleibenden Warteschlangen an.

Wenn Sie keine Befehlszeilenargumente angeben, zeigt die Anwendung einfach eine Liste der vorhandenen Warteschlangen an.

Die folgenden Abschnitte enthalten Auszüge aus diesem Beispiel. Der vollständige Code für das Beispiel wird danach angezeigt und kann unverändert erstellt und ausgeführt werden.

Themen

- Lösche die Warteschlange
- Warten Sie, bis die Warteschlange weg ist
- Zeigt eine Liste vorhandener Warteschlangen
- Vollständiger Code
- Weitere Überlegungen

Lösche die Warteschlange

Das folgende Snippet löscht die Warteschlange, die durch die angegebene Warteschlangen-URL identifiziert wurde.

Das Beispiel am Ende dieses Themas zeigt, wie dieses Snippet verwendet wird.

```
    //
```

```
    // 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.");
    }
```

Warten Sie, bis die Warteschlange weg ist

Das folgende Snippet wartet, bis der Löschvorgang abgeschlossen ist, was 60 Sekunden dauern kann.

Das Beispiel am Ende dieses Themas zeigt, wie dieser Codeausschnitt verwendet wird.

```
    //
    // 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;
      }
    }
```

Zeigt eine Liste vorhandener Warteschlangen

Der folgende Ausschnitt zeigt eine Liste der vorhandenen Warteschlangen in der Region des SQS-Clients.

Das Beispiel am Ende dieses Themas zeigt, wie dieser Codeausschnitt verwendet wird.

```
//
// 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}");
}
```

Vollständiger Code

Dieser Abschnitt enthält relevante Referenzen und den vollständigen Code für dieses Beispiel.

SDK-Referenzen

NuGet Pakete:

- AWSSDK.SQS

Elemente der Programmierung:

- Namespace Amazon.sqs

  Klasse Amazon SQSClient
- Namespace Amazon.SQS.Model

  Klasse ListQueuesResponse

Der Code

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

Weitere Überlegungen

- Der `DeleteQueueAsync` API-Aufruf überprüft nicht, ob die Warteschlange, die Sie löschen, als Warteschlange für unzustellbare Briefe verwendet wird. Ein ausgefeilteres Verfahren könnte dies überprüfen.

- Sie können die Liste der Warteschlangen und die Ergebnisse dieses Beispiels auch in der Amazon SQS SQS-Konsole sehen.

Amazon SQS SQS-Nachrichten senden

Dieses Beispiel zeigt Ihnen, wie Sie Nachrichten AWS SDK für .NET an eine Amazon SQS SQS-Warteschlange senden, die Sie programmgesteuert oder mithilfe der Amazon SQS SQS-Konsole erstellen können. Die Anwendung sendet eine einzelne Nachricht an die Warteschlange und anschließend einen Stapel von Nachrichten. Die Anwendung wartet dann auf Benutzereingaben, bei denen es sich um zusätzliche Nachrichten handeln kann, die an die Warteschlange gesendet werden sollen, oder um eine Aufforderung zum Beenden der Anwendung.

Dieses und das nächste Beispiel für den Empfang von Nachrichten können zusammen verwendet werden, um den Nachrichtenfluss in Amazon SQS zu sehen.

Die folgenden Abschnitte enthalten Auszüge aus diesem Beispiel. Der vollständige Code für das Beispiel wird danach angezeigt und kann unverändert erstellt und ausgeführt werden.

Themen

- Senden einer Nachricht
- Senden Sie einen Stapel von Nachrichten

- Löscht alle Nachrichten aus der Warteschlange
- Vollständiger Code
- Weitere Überlegungen

Senden einer Nachricht

Das folgende Snippet sendet eine Nachricht an die Warteschlange, die durch die angegebene Warteschlangen-URL identifiziert wird.

Das Beispiel am Ende dieses Themas zeigt, wie dieses Snippet verwendet wird.

```
//
// 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}");
}
```

Senden Sie einen Stapel von Nachrichten

Das folgende Snippet sendet einen Stapel von Nachrichten an die Warteschlange, die durch die angegebene Warteschlangen-URL identifiziert wird.

Das Beispiel am Ende dieses Themas zeigt, wie dieses Snippet verwendet wird.

```
//
// 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.");
  }
```

Löscht alle Nachrichten aus der Warteschlange

Das folgende Snippet löscht alle Nachrichten aus der Warteschlange, die durch die angegebene Warteschlangen-URL identifiziert wird. Dies wird auch als Löschen der Warteschlange bezeichnet.

Das Beispiel am Ende dieses Themas zeigt, wie dieser Codeausschnitt verwendet wird.

```
  //
  // 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}");
  }
```

Vollständiger Code

Dieser Abschnitt enthält relevante Referenzen und den vollständigen Code für dieses Beispiel.

SDK-Referenzen

NuGet Pakete:

- AWSSDK.SQS

Elemente der Programmierung:

- Namespace Amazon.sqs

  Klasse Amazon SQSClient
- Namespace Amazon.SQS.Model

  Klasse PurgeQueueResponse

  Klasse SendMessageBatchResponse

  Klasse SendMessageResponse

Klasse SendMessageBatchRequestEntry

Klasse SendMessageBatchResultEntry

Der Code

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon.SQS;
using Amazon.SQS.Model;

namespace SQSSendMessages
{
  // = = =
```

```
    }

    // Create the Amazon SQS client
    var sqsClient = new AmazonSQSClient();

    // (could verify that the queue exists)
    // 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}");
    }
  }
}
```

Weitere Überlegungen

- Informationen zu verschiedenen Beschränkungen für Nachrichten, einschließlich der zulässigen Zeichen, finden Sie unter Kontingente für Nachrichten im Amazon Simple Queue Service Developer Guide.

- Nachrichten bleiben in Warteschlangen, bis sie gelöscht werden oder die Warteschlange gelöscht wird. Wenn eine Nachricht von einer Anwendung empfangen wurde, ist sie in der Warteschlange nicht sichtbar, obwohl sie noch in der Warteschlange vorhanden ist. Weitere Informationen zu Sichtbarkeits-Timeouts finden Sie unter Amazon SQS-Sichtbarkeits-Timeout.

- Zusätzlich zum Nachrichtentext können Sie Nachrichten auch Attribute hinzufügen. Weitere Informationen finden Sie unter Nachrichtenmetadaten.

Empfangen Amazon SQS SQS-Nachrichten

Dieses Beispiel zeigt Ihnen, wie Sie die verwenden, AWS SDK für .NET um Nachrichten aus einer Amazon SQS SQS-Warteschlange zu empfangen, die Sie programmgesteuert oder mithilfe der Amazon SQS SQS-Konsole erstellen können. Die Anwendung liest eine einzelne Nachricht aus der Warteschlange, verarbeitet die Nachricht (in diesem Fall zeigt sie den Nachrichtentext auf der Konsole an) und löscht dann die Nachricht aus der Warteschlange. Die Anwendung wiederholt diese Schritte, bis der Benutzer eine Taste auf der Tastatur eingibt.

Dieses Beispiel und das vorherige Beispiel zum Senden von Nachrichten können zusammen verwendet werden, um den Nachrichtenfluss in Amazon SQS zu sehen.

Die folgenden Abschnitte enthalten Auszüge aus diesem Beispiel. Der vollständige Code für das Beispiel wird danach angezeigt und kann unverändert erstellt und ausgeführt werden.

Themen

- Empfangen Sie eine Nachricht
- Eine Nachricht löschen
- Vollständiger Code
- Weitere Überlegungen

Empfangen Sie eine Nachricht

Das folgende Snippet empfängt eine Nachricht aus der Warteschlange, die durch die angegebene Warteschlangen-URL identifiziert wird.

Das Beispiel am Ende dieses Themas zeigt, wie dieses Snippet verwendet wird.

```
//
// 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.)
  });
}
```

Eine Nachricht löschen

Das folgende Snippet löscht eine Nachricht aus der Warteschlange, die durch die angegebene Warteschlangen-URL identifiziert wird.

Das Beispiel am Ende dieses Themas zeigt, wie dieses Snippet verwendet wird.

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

Vollständiger Code

Dieser Abschnitt enthält relevante Referenzen und den vollständigen Code für dieses Beispiel.

SDK-Referenzen

NuGet Pakete:

- AWSSDK.SQS

Elemente der Programmierung:

- Namespace Amazon.sqs

  Klasse Amazon SQSClient
- Namespace Amazon.SQS.Model

  Klasse ReceiveMessageRequest

  Klasse ReceiveMessageResponse

Der Code

```
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);
    }
  }
}
```

Weitere Überlegungen

- Um Long Polling anzugeben, verwendet dieses Beispiel die `WaitTimeSeconds` Eigenschaft für jeden Aufruf der `ReceiveMessageAsync` Methode.

  Sie können auch lange Abfragen für alle Nachrichten in einer Warteschlange angeben, indem Sie das `ReceiveMessageWaitTimeSeconds` Attribut beim Erstellen oder Aktualisieren der Warteschlange verwenden.

  Informationen zu kurzen Abfragen im Vergleich zu langen Abfragen finden Sie unter Kurze und lange Abfragen im Amazon Simple Queue Service Developer Guide.

- Während der Nachrichtenverarbeitung können Sie die Empfangsnummer verwenden, um das Timeout für die Nachrichtensichtbarkeit zu ändern. Informationen dazu finden Sie in den `ChangeMessageVisibilityAsync` Methoden der SQSClientAmazon-Klasse.

- Durch das `DeleteMessageAsync` bedingungslose Aufrufen der Methode wird die Nachricht unabhängig von der Einstellung für das Sichtbarkeits-Timeout aus der Warteschlange entfernt.

AWS Lambda Für den Rechendienst verwenden

Die AWS SDK für .NET Unterstützung AWS Lambda, mit der Sie Code ausführen können, ohne Server bereitstellen oder verwalten zu müssen. Weitere Informationen finden Sie auf der AWS Lambda Produktseite und im AWS Lambda Entwicklerhandbuch, insbesondere im Abschnitt Arbeiten mit C#.

APIs

Das AWS SDK für .NET sieht vor APIs . AWS Lambda Sie APIs ermöglichen es Ihnen, mit Lambda-Funktionen wie Funktionen, Triggern und Ereignissen zu arbeiten. Den vollständigen Satz von finden Sie APIs unter Lambda in der AWS SDK für .NET API-Referenz.

Die Lambda APIs werden durch NuGetPakete bereitgestellt.

Voraussetzungen

Bevor Sie beginnen, stellen Sie sicher, dass Sie Ihre Umgebung eingerichtet und Ihr Projekt konfiguriert haben. Lesen Sie auch die Informationen unterVerwenden der SDK.

Zusätzliche Informationen

Informationen Integration AWS mit.NET Aspire zur Entwicklung mit.NET Aspire AWS Lambda finden Sie unter.

Themen

Themen

- Verwenden von Anmerkungen zum Schreiben von Funktionen AWS Lambda

Verwenden von Anmerkungen zum Schreiben von Funktionen AWS Lambda

Beim Schreiben von Lambda-Funktionen müssen Sie unter anderem manchmal eine große Menge an Handlercode schreiben und AWS CloudFormation Vorlagen aktualisieren. Lambda Annotations ist ein Framework, das dazu beiträgt, diese Belastung für Lambda-Funktionen von .NET 6 zu verringern und so dafür zu sorgen, dass sich das Schreiben von Lambda in C# natürlicher anfühlt.

Betrachten Sie als Beispiel für die Vorteile der Verwendung des Lambda Annotations Frameworks die folgenden Codefragmente, die zwei Zahlen addieren.

Ohne Lambda-Anmerkungen

```
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" } }
        };
    }
}
```

Mit Lambda-Anmerkungen

```
public class Functions
{
    [LambdaFunction]
```

```
    [RestApi("/plus/{x}/{y}")]
    public int Plus(int x, int y)
    {
        return x + y;
    }
}
```

Wie im Beispiel gezeigt, können Lambda Annotations die Notwendigkeit eines bestimmten Boilerplate-Codes überflüssig machen.

Einzelheiten zur Verwendung des Frameworks sowie weitere Informationen finden Sie in den folgenden Ressourcen:

- Die GitHub README-Datei für die Dokumentation zu den APIs und Attributen von Lambda-Annotationen.
- Blogbeitrag zu Lambda Annotations
- Das `Amazon.Lambda.Annotations` NuGet -Paket.
- Das Photo Asset Management-Projekt am. GitHub Sehen Sie sich insbesondere den PamApiAnnotationsOrdner und die Verweise auf Lambda-Anmerkungen in der README-Datei des Projekts an.

> **Note**
>
> Das vorherige Beispiel ist spezifisch für V3 von. AWS SDK für .NET Wenn Sie das Beispiel mit Version 4 des SDK (der neuesten Version) verwenden, müssen Sie möglicherweise Anpassungen entsprechend den Informationen unter vornehmenMigration auf Version 4.

Hochrangige Bibliotheken und Frameworks für AWS SDK für .NET

Die folgenden Abschnitte enthalten Informationen zu Bibliotheken und Frameworks auf hoher Ebene, die nicht Teil der Kernfunktionen des SDK sind. Diese Bibliotheken und Frameworks verwenden die Kernfunktionen des SDK, um Funktionen zu erstellen, die bestimmte Aufgaben erleichtern.

Wenn Sie mit dem noch nicht vertraut sind AWS SDK für .NET, sollten Sie sich zuerst mit dem Erstellen einer einfachen Anwendung Thema befassen. Es gibt Ihnen eine Einführung in das SDK.

Bevor Sie beginnen, stellen Sie sicher, dass Sie Ihre Umgebung eingerichtet und Ihr Projekt konfiguriert haben. Lesen Sie auch die Informationen unterVerwenden der SDK.

Themen

- AWS Message Processing Framework für.NET
- Integration AWS mit .NET Aspire in AWS SDK für .NET

AWS Message Processing Framework für.NET

Das AWS Message Processing Framework for .NET ist ein AWS natives Framework, das die Entwicklung von.NET-Nachrichtenverarbeitungsanwendungen vereinfacht, die AWS Dienste wie Amazon Simple Queue Service (SQS), Amazon Simple Notification Service (SNS) und Amazon verwenden. EventBridge Das Framework reduziert den Umfang an Standardcode, den Entwickler schreiben müssen, sodass Sie sich bei der Veröffentlichung und Nutzung von Nachrichten auf Ihre Geschäftslogik konzentrieren können. Einzelheiten darüber, wie das Framework Ihre Entwicklung vereinfachen kann, finden Sie im Blogbeitrag Introducing the AWS Message Processing Framework for .NET (Preview). Insbesondere der erste Teil enthält eine Demonstration, die den Unterschied zwischen der Verwendung von API-Aufrufen auf niedriger Ebene und der Verwendung des Frameworks zeigt.

Das Message Processing Framework unterstützt die folgenden Aktivitäten und Funktionen:

- Senden von Nachrichten an SQS und Veröffentlichen von Ereignissen in SNS und. EventBridge
- Empfangen und Verarbeiten von Nachrichten von SQS mithilfe eines Pollers mit langer Laufzeit, der normalerweise in Hintergrunddiensten verwendet wird. Dazu gehört die Verwaltung des Sichtbarkeits-Timeouts während der Bearbeitung einer Nachricht, um zu verhindern, dass andere Clients sie verarbeiten.
- Umgang mit Nachrichten in AWS Lambda Funktionen.
- FIFO (first-in-first-out) SQS-Warteschlangen und SNS-Themen.
- OpenTelemetry für die Protokollierung.

Einzelheiten zu diesen Aktivitäten und Funktionen finden Sie im Abschnitt Funktionen des Blogbeitrags und in den unten aufgeführten Themen.

Bevor Sie beginnen, stellen Sie sicher, dass Sie Ihre Umgebung eingerichtet und Ihr Projekt konfiguriert haben. Lesen Sie auch die Informationen unterVerwenden der SDK.

Weitere Ressourcen

- Das `AWS.Messaging`Paket auf NuGet.org.

- Die API-Referenz.
- Die `README` Datei im GitHub Repo unter https://github.com/aws/aws-dotnet-messaging/
- .NET-Abhängigkeitsinjektion von Microsoft.
- .NET Generic Host von Microsoft.

Themen

- Erste Schritte mit dem AWS Message Processing Framework für.NET
- Veröffentlichen Sie Nachrichten mit dem AWS Message Processing Framework for .NET
- Nachrichten mit dem AWS Message Processing Framework for .NET konsumieren
- Verwenden von FIFO mit dem AWS Message Processing Framework für.NET
- Logging und offene Telemetrie für das AWS Message Processing Framework für.NET
- Das AWS Message Processing Framework für.NET anpassen
- Sicherheit für das AWS Message Processing Framework für.NET

Erste Schritte mit dem AWS Message Processing Framework für.NET

Bevor Sie beginnen, stellen Sie sicher, dass Sie Ihre Umgebung eingerichtet und Ihr Projekt konfiguriert haben. Lesen Sie auch die Informationen unterVerwenden der SDK.

Dieses Thema enthält Informationen, die Ihnen den Einstieg in die Verwendung des Message Processing Framework erleichtern. Zusätzlich zu den Voraussetzungen und zur Konfiguration wird ein Tutorial bereitgestellt, das Ihnen zeigt, wie Sie ein gängiges Szenario implementieren.

Voraussetzungen und Konfiguration

- Die Anmeldeinformationen, die Sie für Ihre Anwendung angeben, müssen über die entsprechenden Berechtigungen für den Messaging-Dienst und die von ihr verwendeten Vorgänge verfügen. Weitere Informationen finden Sie in den Sicherheitsthemen für SQS und SNS sowie EventBridgein den entsprechenden Entwicklerhandbüchern. Lesen Sie auch den Teil der README-Datei, in dem bestimmte Berechtigungen GitHub beschrieben werden.
- Um das AWS Message Processing Framework für.NET verwenden zu können, müssen Sie das `AWS.Messaging` NuGetPaket zu Ihrem Projekt hinzufügen. Zum Beispiel:

```
dotnet add package AWS.Messaging
```

- Das Framework lässt sich in den Dienstcontainer Dependency Injection (DI) von .NET integrieren. Sie können das Framework beim Start Ihrer Anwendung konfigurieren, indem Sie es aufrufen`AddAWSMessageBus`, um es dem DI-Container hinzuzufügen.

```
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

Dieses Tutorial zeigt, wie Sie das AWS Message Processing Framework für.NET verwenden. Es erstellt zwei Anwendungen: eine ASP.NET Core Minimal API, die Nachrichten an eine Amazon SQS SQS-Warteschlange sendet, wenn sie eine Anfrage an einem API-Endpunkt empfängt, und eine Konsolenanwendung mit langer Laufzeit, die nach diesen Nachrichten fragt und sie verarbeitet.

- Die Anweisungen in diesem Tutorial bevorzugen die .NET-CLI, aber Sie können dieses Tutorial entweder mit plattformübergreifenden Tools wie dem.NET-CLI oder Microsoft Visual Studio ausführen. Informationen zu Tools finden Sie unterInstallation und Konfiguration Ihrer Toolchain für die AWS SDK für .NET.
- In dieser Anleitung wird davon ausgegangen, dass Sie Ihr `[default]` Profil für Anmeldeinformationen verwenden. Es wird auch davon ausgegangen, dass kurzfristige Anmeldeinformationen mit entsprechenden Berechtigungen für das Senden und Empfangen von Amazon SQS SQS-Nachrichten verfügbar sind. Weitere Informationen finden Sie unter Authentifizieren des mit AWS SDK für .NETAWS und in den Sicherheitsthemen für SQS.

> **Note**
>
> Wenn Sie dieses Tutorial ausführen, können Ihnen Kosten für SQS-Messaging entstehen.

Schritte

- Erstellen Sie eine SQS-Warteschlange
- Erstellen Sie die Veröffentlichungsanwendung und führen Sie sie aus
- Erstellen Sie die Bearbeitungsanwendung und führen Sie sie aus
- Bereinigen

Erstellen Sie eine SQS-Warteschlange

Für dieses Tutorial ist eine SQS-Warteschlange erforderlich, an die Nachrichten gesendet und von der Nachrichten empfangen werden können. Eine Warteschlange kann erstellt werden, indem Sie einen der folgenden Befehle für AWS CLI oder für verwenden. AWS -Tools für PowerShell Notieren Sie sich die zurückgegebene Warteschlangen-URL, damit Sie sie in der folgenden Framework-Konfiguration angeben können.

AWS CLI

```
aws sqs create-queue --queue-name DemoQueue
```

AWS -Tools für PowerShell

```
New-SQSQueue -QueueName DemoQueue
```

Erstellen Sie die Veröffentlichungsanwendung und führen Sie sie aus

Gehen Sie wie folgt vor, um die Veröffentlichungsanwendung zu erstellen und auszuführen.

1. Öffnen Sie eine Befehlszeile oder ein Terminal. Suchen oder erstellen Sie einen Betriebssystemordner, unter dem Sie ein .NET-Projekt erstellen können.
2. Führen Sie in diesem Ordner den folgenden Befehl aus, um das .NET-Projekt zu erstellen.

   ```
   dotnet new webapi --name Publisher
   ```

3. Navigieren Sie in den Ordner des neuen Projekts. Fügen Sie eine Abhängigkeit vom AWS Message Processing Framework for .NET hinzu.

   ```
   cd Publisher
   ```

```
dotnet add package AWS.Messaging
```

> **Note**
>
> Wenn Sie AWS IAM Identity Center für die Authentifizierung verwenden, achten Sie darauf, auch `AWSSDK.SSO` und hinzuzufügen`AWSSDK.SSOOIDC`.

4. Ersetzen Sie den Code in `Program.cs` durch den folgenden Code.

```
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. Führen Sie den folgenden Befehl aus. Dadurch sollte ein Browserfenster mit der Swagger-Benutzeroberfläche geöffnet werden, mit der Sie Ihre API erkunden und testen können.

```
dotnet watch run <queue URL created earlier>
```

6. Öffnen Sie den `/greeting` Endpunkt und wählen Sie Ausprobieren.
7. Geben Sie `senderName` `greeting` Werte für die Nachricht an und wählen Sie Ausführen. Dadurch wird Ihre API aufgerufen, die die SQS-Nachricht sendet.

Erstellen Sie die Bearbeitungsanwendung und führen Sie sie aus

Gehen Sie wie folgt vor, um die Bearbeitungsanwendung zu erstellen und auszuführen.

1. Öffnen Sie eine Befehlszeile oder ein Terminal. Suchen oder erstellen Sie einen Betriebssystemordner, unter dem Sie ein .NET-Projekt erstellen können.
2. Führen Sie in diesem Ordner den folgenden Befehl aus, um das .NET-Projekt zu erstellen.

```
dotnet new console --name Handler
```

3. Navigieren Sie in den Ordner des neuen Projekts. Fügen Sie eine Abhängigkeit vom AWS Message Processing Framework for .NET hinzu. Fügen Sie außerdem das `Microsoft.Extensions.Hosting` Paket hinzu, mit dem Sie das Framework über den.NET Generic Host konfigurieren können.

```
cd Handler
dotnet add package AWS.Messaging
dotnet add package Microsoft.Extensions.Hosting
```

> **Note**
>
> Wenn Sie AWS IAM Identity Center für die Authentifizierung verwenden, stellen Sie sicher, dass Sie auch `AWSSDK.SSO` und hinzufügen`AWSSDK.SSOOIDC`.

4. Ersetzen Sie den Code in `Program.cs` durch den folgenden Code.

```
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. Führen Sie den folgenden Befehl aus. Dadurch wird ein Poller mit langer Laufzeit gestartet.

```
dotnet run <queue URL created earlier>
```

Kurz nach dem Start empfängt die Anwendung die Nachricht, die im ersten Teil dieses Tutorials gesendet wurde, und protokolliert die folgende Meldung:

```
Received message {greeting} from {senderName}
```

6. Drücken Sie`Ctrl+C`, um den Poller zu stoppen.

Bereinigen

Verwenden Sie einen der folgenden Befehle für AWS CLI oder, AWS -Tools für PowerShell um die Warteschlange zu löschen.

AWS CLI

```
aws sqs delete-queue --queue-url "<queue URL created earlier>"
```

AWS -Tools für PowerShell

```
Remove-SQSQueue -QueueUrl "<queue URL created earlier>"
```

Veröffentlichen Sie Nachrichten mit dem AWS Message Processing Framework for .NET

Das AWS Message Processing Framework for .NET unterstützt das Veröffentlichen eines oder mehrerer Nachrichtentypen, das Verarbeiten eines oder mehrerer Nachrichtentypen oder beides in derselben Anwendung.

Der folgende Code zeigt eine Konfiguration für eine Anwendung, die verschiedene Nachrichtentypen für verschiedene AWS Dienste veröffentlicht.

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

Nachdem Sie das Framework beim Start registriert haben, fügen Sie das Generikum `IMessagePublisher` in Ihren Code ein. Rufen Sie seine `PublishAsync` Methode auf, um einen der oben konfigurierten Nachrichtentypen zu veröffentlichen. Der generische Herausgeber bestimmt das Ziel, an das die Nachricht weitergeleitet werden soll, anhand ihres Typs.

Im folgenden Beispiel empfängt ein ASP.NET-MVC-Controller sowohl `ChatMessage` Nachrichten als auch `OrderInfo` Ereignisse von Benutzern und veröffentlicht sie dann in Amazon SQS bzw. Amazon SNS. Beide Nachrichtentypen können mit dem generischen Herausgeber veröffentlicht werden, der oben konfiguriert wurde.

```
[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();
    }
}
```

Um eine Nachricht an die entsprechende Verarbeitungslogik weiterzuleiten, verwendet das Framework Metadaten, die als Nachrichtentyp-ID bezeichnet werden. Standardmäßig ist dies der vollständige Name des.NET-Typs der Nachricht, einschließlich des Assemblynamens. Wenn Sie Nachrichten sowohl senden als auch bearbeiten, funktioniert dieser Mechanismus gut, wenn Sie die Definition Ihrer Nachrichtenobjekte projektübergreifend gemeinsam verwenden. Wenn die Nachrichten jedoch in verschiedenen Namespaces neu definiert werden oder wenn Sie Nachrichten mit anderen Frameworks oder Programmiersprachen austauschen, müssen Sie möglicherweise die ID des Nachrichtentyps überschreiben.

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

Dienstspezifische Herausgeber

Das oben gezeigte Beispiel verwendet das Generikum`IMessagePublisher`, das basierend auf dem konfigurierten Nachrichtentyp in jedem unterstützten AWS Dienst veröffentlichen kann. Das Framework bietet auch servicespezifische Herausgeber für Amazon SQS, Amazon SNS und

Amazon. EventBridge Diese speziellen Herausgeber stellen Optionen zur Verfügung, die nur für diesen Service gelten und mit den Typen`ISQSPublisher`, `ISNSPublisher` und eingegeben werden können. `IEventBridgePublisher`

Wenn Sie beispielsweise Nachrichten an eine SQS-FIFO-Warteschlange senden, müssen Sie die entsprechende Nachrichtengruppen-ID festlegen. Der folgende Code zeigt das `ChatMessage` Beispiel erneut, verwendet aber jetzt eine, `ISQSPublisher` um SQS-spezifische Optionen festzulegen.

```
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();
```

```
    }
}
```

Dasselbe kann für SNS und jeweils mit `ISNSPublisher` und EventBridge gemacht werden. `IEventBridgePublisher`

```
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>
});
```

Standardmäßig werden Nachrichten eines bestimmten Typs an das Ziel gesendet, das im Voraus konfiguriert wurde. Sie können das Ziel für eine einzelne Nachricht jedoch mithilfe der nachrichtenspezifischen Herausgeber außer Kraft setzen. Sie können auch den zugrundeliegenden AWS SDK für .NET Client überschreiben, der zum Veröffentlichen der Nachricht verwendet wird. Dies kann in Mehrmandantenanwendungen nützlich sein, bei denen Sie je nach Ziel Rollen oder Anmeldeinformationen ändern müssen.

```
await _sqsPublisher.SendAsync(message, new SQSOptions
{
    OverrideClient = <override IAmazonSQS client>,
    QueueUrl = <override queue URL>
});
```

Nachrichten mit dem AWS Message Processing Framework for .NET konsumieren

Das AWS Message Processing Framework für.NET ermöglicht es Ihnen, Nachrichten zu verarbeiten, die mithilfe des Frameworks oder eines der Messaging-Dienste veröffentlicht wurden. Die

Nachrichten können auf verschiedene Arten konsumiert werden, von denen einige im Folgenden beschrieben werden.

Nachrichten-Handler

Um Nachrichten zu verarbeiten, implementieren Sie einen Message-Handler, der die `IMessageHandler` Schnittstelle für jeden Nachrichtentyp verwendet, den Sie verarbeiten möchten. Die Zuordnung zwischen Nachrichtentypen und Nachrichtenhandlern wird beim Start des Projekts konfiguriert.

```
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();
```

Der folgende Code zeigt ein Beispiel für einen Nachrichtenhandler für eine `ChatMessage` Nachricht.

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

Der äußere `MessageEnvelope` enthält Metadaten, die vom Framework verwendet werden. Seine `message` Eigenschaft ist der Nachrichtentyp (in diesem Fall`ChatMessage`).

Sie können zurückkehren, `MessageProcessStatus.Success()` um anzugeben, dass die Nachricht erfolgreich verarbeitet wurde und das Framework die Nachricht aus der Amazon SQS SQS-Warteschlange löscht. Bei der Rücksendung verbleibt die Nachricht in der Warteschlange`MessageProcessStatus.Failed()`, wo sie erneut verarbeitet oder, sofern konfiguriert, in eine Warteschlange für unzustellbare Briefe verschoben werden kann.

Umgang mit Nachrichten in einem lang andauernden Prozess

Sie können `AddSQSPoller` mit einer URL für die SQS-Warteschlange aufrufen, um einen Long-Running zu starten, der kontinuierlich `BackgroundService`die Warteschlange abfragt und Nachrichten verarbeitet.

```
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();
```

Den SQS Message Poller konfigurieren

Der SQS-Nachrichtenabfrage kann beim Aufrufen konfiguriert werden. `SQSMessagePollerOptions` `AddSQSPoller`

- `MaxNumberOfConcurrentMessages`- Die maximale Anzahl von Nachrichten aus der Warteschlange, die gleichzeitig verarbeitet werden sollen. Der Standardwert lautet 10.
- `WaitTimeSeconds`- Die Dauer (in Sekunden), für die der `ReceiveMessage` SQS-Aufruf darauf wartet, dass eine Nachricht in der Warteschlange eintrifft, bevor er zurückkehrt. Wenn eine Nachricht verfügbar ist, kehrt der Anruf früher als zurück. `WaitTimeSeconds` Der Standardwert ist 20.

Behandlung von Timeouts bei der Sichtbarkeit von Nachrichten

SQS-Nachrichten haben ein Zeitlimit für die Sichtbarkeit. Wenn ein Verbraucher mit der Bearbeitung einer bestimmten Nachricht beginnt, verbleibt sie in der Warteschlange, wird aber vor anderen Verbrauchern verborgen, um zu vermeiden, dass sie mehr als einmal verarbeitet wird. Wenn die Nachricht nicht bearbeitet und gelöscht wird, bevor sie wieder sichtbar wird, versucht möglicherweise ein anderer Verbraucher, dieselbe Nachricht zu bearbeiten.

Das Framework verfolgt Nachrichten, die es gerade bearbeitet, und versucht, das Sichtbarkeits-Timeout zu verlängern. Sie können dieses Verhalten `SQSMessagePollerOptions` beim Aufrufen `AddSQSPoller` konfigurieren.

- `VisibilityTimeout`- Die Dauer in Sekunden, für die empfangene Nachrichten vor nachfolgenden Abrufanfragen verborgen sind. Der Standardwert lautet 30.
- `VisibilityTimeoutExtensionThreshold`- Wenn das Sichtbarkeits-Timeout einer Nachricht innerhalb dieser Sekunden abläuft, verlängert das Framework das Sichtbarkeits-Timeout (um weitere `VisibilityTimeout` Sekunden). Der Standardwert ist 5.
- `VisibilityTimeoutExtensionHeartbeatInterval`- Gibt an, wie oft (in Sekunden) das Framework nach Nachrichten sucht, deren Ablauffrist innerhalb von `VisibilityTimeoutExtensionThreshold` Sekunden abläuft, und dann deren Sichtbarkeits-Timeout verlängert. Der Standardwert lautet 1.

Im folgenden Beispiel sucht das Framework alle 1 Sekunde nach Nachrichten, die noch bearbeitet werden. Für Nachrichten innerhalb von 5 Sekunden, nachdem sie wieder sichtbar werden, verlängert das Framework automatisch das Sichtbarkeits-Timeout jeder Nachricht um weitere 30 Sekunden.

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

Umgang mit Nachrichten in Funktionen AWS Lambda

Sie können das AWS Message Processing Framework für.NET mit der Integration von SQS mit Lambda verwenden. Dies wird durch das Paket bereitgestellt. `AWS.Messaging.Lambda` Lesen Sie die README-Datei, um loszulegen.

Verwenden von FIFO mit dem AWS Message Processing Framework für.NET

Für Anwendungsfälle, in denen Nachrichtenreihenfolge und Nachrichtendeduplizierung entscheidend sind, unterstützt das AWS Message Processing Framework for .NET Amazon SQS SQS-Warteschlangen first-in-first-out (FIFO) und Amazon SNS SNS-Themen.

Veröffentlichen

Wenn Sie Nachrichten in einer FIFO-Warteschlange oder einem FIFO-Thema veröffentlichen, müssen Sie die Nachrichtengruppen-ID festlegen, die die Gruppe angibt, zu der die Nachricht gehört.

Nachrichten innerhalb einer Gruppe werden der Reihe nach verarbeitet. Sie können dies für die SQS-spezifischen und SNS-spezifischen Nachrichtenherausgeber festlegen.

```
await _sqsPublisher.PublishAsync(message, new SQSOptions
{
    MessageDeduplicationId = <message-deduplication-id>,
    MessageGroupId = <message-group-id>
});
```

Abonnieren

Bei der Verarbeitung von Nachrichten aus einer FIFO-Warteschlange behandelt das Framework Nachrichten innerhalb einer bestimmten Nachrichtengruppe in der Reihenfolge, in der sie bei jedem Anruf empfangen wurden. `ReceiveMessages` Das Framework wechselt automatisch in diesen Betriebsmodus, wenn es mit einer Warteschlange konfiguriert ist, die mit endet. `.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();
```

Logging und offene Telemetrie für das AWS Message Processing Framework für.NET

Das AWS Message Processing Framework für.NET ist so konzipiert, dass OpenTelemetry es Ablaufverfolgungen für jede Nachricht protokolliert, die vom Framework veröffentlicht oder verarbeitet wird. Dies wird durch das `AWS.Messaging.Telemetry.OpenTelemetry`Paket bereitgestellt. Lesen Sie die README-Datei, um loszulegen.

> **Note**
>
> Sicherheitsinformationen im Zusammenhang mit der Protokollierung finden Sie unterSicherheit für das AWS Message Processing Framework für.NET.

Das AWS Message Processing Framework für.NET anpassen

Das AWS Message Processing Framework für.NET erstellt, sendet und verarbeitet Nachrichten in drei verschiedenen „Ebenen“:

1. Auf der äußersten Ebene erstellt das Framework die AWS-native Anfrage oder Antwort, die für einen Dienst spezifisch ist. Mit Amazon SQS erstellt es beispielsweise `SendMessage`Anfragen und arbeitet mit den `Message`Objekten, die durch den Service definiert sind.
2. In der SQS-Anfrage und -Antwort setzt das Framework das `MessageBody` Element (oder `Message` für Amazon SNS oder `Detail` für Amazon EventBridge) auf ein JSON-Format. CloudEvent Dies enthält vom Framework festgelegte Metadaten, auf die bei der Bearbeitung einer Nachricht über das `MessageEnvelope` Objekt zugegriffen werden kann.
3. Auf der innersten Ebene enthält das `data` Attribut innerhalb des CloudEvent JSON-Objekts eine JSON-Serialisierung des .NET-Objekts, das als Nachricht gesendet oder empfangen wurde.

```
{
    "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>"
}
```

Sie können anpassen, wie der Nachrichtenumschlag konfiguriert und gelesen wird:

- `"id"`identifiziert die Nachricht eindeutig. Standardmäßig ist es auf eine neue GUID gesetzt, aber dies kann überschrieben werden, indem Sie Ihre eigene GUID implementieren `IMessageIdGenerator` und diese in den DI-Container einfügen.
- `"type"`steuert, wie die Nachricht an die Handler weitergeleitet wird. Standardmäßig wird dabei der vollständige Name des.NET-Typs verwendet, der der Nachricht entspricht. Sie können dies über

den `messageTypeIdentifier` Parameter überschreiben, wenn Sie den Nachrichtentyp über `AddSQSPublisher` `AddSNSPublisher`, oder dem Ziel zuordnen `AddEventBridgePublisher`.

- `"source"` gibt an, welches System oder welcher Server die Nachricht gesendet hat.
  - Dies ist der Funktionsname bei der Veröffentlichung von AWS Lambda, der Clustername und der Task-ARN bei Amazon ECS, die Instance-ID bei Amazon EC2, andernfalls ein Fallback-Wert von `/aws/messaging`.
  - Sie können dies über `AddMessageSource` oder `AddMessageSourceSuffix` auf dem `MessageBusBuilder` überschreiben.
- `"time"` auf den aktuellen Wert DateTime in UTC gesetzt. Dies kann außer Kraft gesetzt werden, indem Sie Ihr eigenes implementieren `IDateTimeHandler` und dieses in den DI-Container einfügen.
- `"data"` enthält eine JSON-Darstellung des .NET-Objekts, das als Nachricht gesendet oder empfangen wurde:
  - `ConfigureSerializationOptions` on `MessageBusBuilder` ermöglicht es Ihnen, das zu konfigurieren `System.Text.Json.JsonSerializerOptions`, das beim Serialisieren und Deserialisieren der Nachricht verwendet wird.
  - Um zusätzliche Attribute einzufügen oder den Nachrichtenumschlag zu transformieren, sobald das Framework ihn erstellt hat, können Sie dies über on implementieren `ISerializationCallback` und registrieren. `AddSerializationCallback` `MessageBusBuilder`

Sicherheit für das AWS Message Processing Framework für.NET

Das AWS Message Processing Framework für .NET basiert auf der AWS SDK für .NET für die Kommunikation mit AWS. Weitere Hinweise zur Sicherheit finden Sie AWS SDK für .NET unterSicherheit für dieses AWS Produkt oder diesen Service.

Aus Sicherheitsgründen protokolliert das Framework keine vom Benutzer gesendeten Datennachrichten. Wenn Sie diese Funktionalität zu Debugging-Zwecken aktivieren möchten, müssen Sie den Message Bus wie folgt aufrufen`EnableDataMessageLogging()`:

```
builder.Services.AddAWSMessageBus(bus =>
{
    builder.EnableDataMessageLogging();
});
```

Wenn Sie ein potenzielles Sicherheitsproblem entdecken, finden Sie Informationen zur Meldung in der Sicherheitsrichtlinie.

Integration AWS mit .NET Aspire in AWS SDK für .NET

.NET Aspire ist eine neue Methode zur Erstellung cloudfähiger Anwendungen. Insbesondere bietet es eine Orchestrierung für lokale Umgebungen, in denen die Komponenten verteilter Anwendungen ausgeführt, verbunden und debuggt werden können. Um die interne Entwicklungsschleife für cloudfähige Anwendungen zu verbessern, wurden Integrationen mit .NET Aspire entwickelt, um Ihre .NET-Anwendungen mit Ressourcen zu verbinden. AWS Diese Integrationen sind über das NuGet Aspire.Hosting.AWS-Paket verfügbar.

Die folgenden.NET Aspire-Integrationen sind verfügbar:

- Die Fähigkeit, Ihre AWS Ressourcen bereitzustellen über AWS CloudFormation. Diese Integration wird im .NET AppHost Aspire-Projekt verwendet.

  Weitere Informationen finden Sie im Blogbeitrag Integration AWS mit.NET Aspire.
- Lokales Installieren, Konfigurieren und Herstellen einer Verbindung AWS SDK für .NET zu Amazon DynamoDB. Diese Integration wird im .NET AppHost Aspire-Projekt verwendet.

  Weitere Informationen finden Sie im Blogbeitrag Integration AWS mit.NET Aspire.
- Aktivieren Sie eine lokale Entwicklungsumgebung für AWS LambdaFunktionen. Diese Integration wird im .NET AppHost Aspire-Projekt verwendet.

  Weitere Informationen finden Sie in den Blogbeiträgen Erstellen und Debuggen von.NET Lambda-Anwendungen mit.NET Aspire (Teil 1) und Erstellen und Debuggen von.NET Lambda-Anwendungen mit.NET Aspire (Teil 2).

  > **Note**
  >
  > Dies ist die Vorabdokumentation eines Features, das als Vorversion vorliegt. Änderungen sind vorbehalten.

  Da es sich bei dieser Funktion um eine Vorschauversion handelt, müssen Sie sich für die Vorschaufunktionen anmelden. Weitere Informationen zu dieser Vorschaufunktion und dazu, wie Sie sich anmelden können, finden Sie in der Ausgabe zum Entwicklungs-Tracker unter. GitHub

Zusätzliche Informationen

Weitere Informationen und Details dazu, wie Sie die in Aspire.Hosting.aws verfügbaren Integrationen verwenden können, finden Sie in den folgenden Ressourcen.

- Blogbeitrag Integration mit.NET Aspire. AWS
- Blogbeiträge Erstellen und Debuggen von.NET Lambda-Anwendungen mit.NET Aspire (Teil 1) und Erstellen und Debuggen von.NET Lambda-Anwendungen mit.NET Aspire (Teil 2).
- Das integrations-on-dotnet-aspire -for-aws-Repo ist aktiviert. GitHub
- Die ausführliche README-Datei für das Aspire.Hosting.AWS-Paket. NuGet

Support für andere AWS Dienste und Konfigurationen

Der AWS SDK für .NET unterstützt zusätzlich zu den in den vorherigen Abschnitten beschriebenen AWS Diensten. Informationen zu allen unterstützten Diensten finden Sie in der AWS SDK für .NET API-Referenz. APIs

Zusätzlich zu den Namespaces für einzelne AWS Dienste bietet der AWS SDK für .NET auch Folgendes: APIs

Flächen	Beschreibung	Ressourcen
AWS Support	Programmgesteuerter Zugriff auf AWS Support-Fälle und Trusted Advisor Advisor-Funktionen.	Siehe Amazon. AWSSupport und Amazon. AWSSupport. Modell.
Allgemeines	Helfer-Klassen und Aufzählungen.	Weitere Informationen finden Sie unter Amazon und Amazon.Util.

SDK für .NET (v4) Codebeispiele

Die Codebeispiele in diesem Thema zeigen Ihnen, wie Sie AWS SDK für .NET (v4) mit verwenden AWS.

Bei Grundlagen handelt es sich um Code-Beispiele, die Ihnen zeigen, wie Sie die wesentlichen Vorgänge innerhalb eines Services ausführen.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Szenarien sind Code-Beispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb eines Services aufrufen oder mit anderen AWS-Services kombinieren.

Einige Dienste enthalten zusätzliche Beispielkategorien, die zeigen, wie Sie Bibliotheken oder Funktionen nutzen können, die für den Dienst spezifisch sind.

Services

- Aurora-Beispiele mit SDK für .NET (v4)
- Beispiele für Auto Scaling mit SDK für .NET (v4)
- Amazon Bedrock — Beispiele für die Verwendung von SDK für .NET (v4)
- Amazon Bedrock Runtime-Beispiele mit SDK für .NET (v4)
- AWS CloudFormation Beispiele mit SDK für .NET (v4)
- CloudWatch Beispiele mit SDK für .NET (v4)
- Beispiele für Amazon Cognito Identity Provider mit SDK für .NET (v4)
- AWS Control Tower Beispiele mit SDK für .NET (v4)
- DynamoDB-Beispiele mit SDK für .NET (v4)
- EC2 Amazon-Beispiele mit SDK für .NET (v4)
- Amazon ECS-Beispiele mit SDK für .NET (v4)
- Amazon S3 S3-Beispiele mit SDK für .NET (v4)

Aurora-Beispiele mit SDK für .NET (v4)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK für .NET (v4) mit Aurora Aktionen ausführen und allgemeine Szenarien implementieren.

Bei Grundlagen handelt es sich um Code-Beispiele, die Ihnen zeigen, wie Sie die wesentlichen Vorgänge innerhalb eines Services ausführen.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Erste Schritte

Hello Aurora

Die folgenden Codebeispiele veranschaulichen die ersten Schritte mit Aurora.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
using 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}.");
            }
        }
    }
}
```

- Einzelheiten zur API finden Sie unter Describe DBClusters in der AWS SDK für .NET API-Referenz.

Themen

- Grundlagen
- Aktionen

Grundlagen

Erlernen der Grundlagen

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Erstellen Sie eine benutzerdefinierte Aurora-DB-Cluster-Parametergruppe und legen Sie Parameterwerte fest.
- Erstellen Sie einen DB-Cluster, der die Parametergruppe verwendet.
- Erstellen Sie eine DB-Instance, die eine Datenbank enthält.
- Erstellen Sie einen Snapshot des DB-Clusters und bereinigen Sie dann die Ressourcen.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Führen Sie ein interaktives Szenario an einem Prompt aus.

```
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;
    }
```

Wrapper-Methoden, die vom Szenario aufgerufen werden, um Aurora-Aktionen zu verwalten.

```
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;
    }
}
```

- API-Details finden Sie in den folgenden Themen der AWS SDK für .NET -API-Referenz.
  - CreateDBCluster
  - CreateDBClusterParameterGroup
  - DBClusterSchnappschuss erstellen
  - CreateDBInstance
  - LöschenDBCluster
  - LöschenDBClusterParameterGroup
  - LöschenDBInstance
  - Beschreiben DBCluster ParameterGroups
  - Beschreiben Sie die DBCluster Parameter
  - Beschreiben Sie DBCluster Schnappschüsse
  - Beschreiben DBClusters
  - DBEngineVersionen beschreiben
  - Beschreiben DBInstances
  - DescribeOrderableDBInstanceOptionen
  - Modifizieren SieDBClusterParameterGroup

Aktionen

`CreateDBCluster`

Das folgende Codebeispiel zeigt die Verwendung`CreateDBCluster`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
/// <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;
}
```

- Einzelheiten zur API finden Sie unter DBCluster In der AWS SDK für .NET API-Referenz erstellen.

`CreateDBClusterParameterGroup`

Das folgende Codebeispiel zeigt die Verwendung`CreateDBClusterParameterGroup`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <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;
    }
```

- Einzelheiten zur API finden Sie unter DBCluster ParameterGroup In der AWS SDK für .NET API-Referenz erstellen.

`CreateDBClusterSnapshot`

Das folgende Codebeispiel zeigt die Verwendung`CreateDBClusterSnapshot`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
   /// <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;
   }
```

- Einzelheiten zur API finden Sie unter DBClusterSnapshot erstellen in der AWS SDK für .NET API-Referenz.

`CreateDBInstance`

Das folgende Codebeispiel zeigt die Verwendung`CreateDBInstance`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <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;
```

```
    }
```

- Einzelheiten zur API finden Sie unter DBInstance In der AWS SDK für .NET API-Referenz erstellen.

`DeleteDBCluster`

Das folgende Codebeispiel zeigt die Verwendung`DeleteDBCluster`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
   /// <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;
   }
```

- Einzelheiten zur API finden Sie unter Löschen DBCluster in der AWS SDK für .NET API-Referenz.

`DeleteDBClusterParameterGroup`

Das folgende Codebeispiel zeigt die Verwendung`DeleteDBClusterParameterGroup`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
/// <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;
}
```

- Einzelheiten zur API finden Sie unter Löschen DBCluster ParameterGroup in der AWS SDK für .NET API-Referenz.

`DeleteDBInstance`

Das folgende Codebeispiel zeigt die Verwendung`DeleteDBInstance`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <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;
    }
```

- Einzelheiten zur API finden Sie unter Löschen DBInstance in der AWS SDK für .NET API-Referenz.

`DescribeDBClusterParameterGroups`

Das folgende Codebeispiel zeigt die Verwendung`DescribeDBClusterParameterGroups`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <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();
    }
```

- Einzelheiten zur API finden Sie unter Describe DBCluster ParameterGroups in der AWS SDK für .NET API-Referenz.

`DescribeDBClusterParameters`

Das folgende Codebeispiel zeigt die Verwendung`DescribeDBClusterParameters`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <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;
    }
```

- Einzelheiten zur API finden Sie unter Describe DBCluster Parameters in der AWS SDK für .NET API-Referenz.

`DescribeDBClusterSnapshots`

Das folgende Codebeispiel zeigt die Verwendung`DescribeDBClusterSnapshots`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <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;
    }
```

- Einzelheiten zur API finden Sie unter Beschreiben von DBCluster Snapshots in der AWS SDK für .NET API-Referenz.

`DescribeDBClusters`

Das folgende Codebeispiel zeigt die Verwendung`DescribeDBClusters`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <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;
    }
```

- Einzelheiten zur API finden Sie unter Describe DBClusters in der AWS SDK für .NET API-Referenz.

`DescribeDBEngineVersions`

Das folgende Codebeispiel zeigt die Verwendung`DescribeDBEngineVersions`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <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;
    }
```

- Einzelheiten zur API finden Sie unter DBEngineVersionen beschreiben in der AWS SDK für .NET API-Referenz.

`DescribeDBInstances`

Das folgende Codebeispiel zeigt die Verwendung`DescribeDBInstances`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
   /// <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;
   }
```

- Einzelheiten zur API finden Sie unter Describe DBInstances in der AWS SDK für .NET API-Referenz.

`DescribeOrderableDBInstanceOptions`

Das folgende Codebeispiel zeigt die Verwendung`DescribeOrderableDBInstanceOptions`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
   /// <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;
   }
```

- Einzelheiten zur API finden Sie unter DescribeOrderableDBInstanceOptionen in der AWS SDK für .NET API-Referenz.

`ModifyDBClusterParameterGroup`

Das folgende Codebeispiel zeigt die Verwendung`ModifyDBClusterParameterGroup`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
   /// <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;
    }
```

- Einzelheiten zur API finden Sie unter Ändern DBCluster ParameterGroup in der AWS SDK für .NET API-Referenz.

Beispiele für Auto Scaling mit SDK für .NET (v4)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK für .NET (v4) mit Auto Scaling Aktionen ausführen und allgemeine Szenarien implementieren.

Bei Grundlagen handelt es sich um Code-Beispiele, die Ihnen zeigen, wie Sie die wesentlichen Vorgänge innerhalb eines Services ausführen.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Erste Schritte

Hallo Auto Scaling

Die folgenden Codebeispiele zeigen, wie Sie mit Auto Scaling beginnen können.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
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
        });
```

```
    }
}
```

- Einzelheiten zur API finden Sie DescribeAutoScalingGroupsin der AWS SDK für .NET API-Referenz.

Themen

- Grundlagen
- Aktionen

Grundlagen

Erlernen der Grundlagen

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Erstellen Sie eine Amazon EC2 Auto Scaling Scaling-Gruppe mit einer Startvorlage und Availability Zones und erhalten Sie Informationen über laufende Instances.
- Aktivieren Sie die Erfassung von CloudWatch Amazon-Metriken.
- Aktualisieren Sie die gewünschte Kapazität der Gruppe und warten Sie, bis eine Instance gestartet wird.
- Beenden Sie eine Instanz in der Gruppe.
- Listet Skalierungsaktivitäten auf, die als Reaktion auf Benutzeranfragen und Kapazitätsänderungen erfolgen.
- Holen Sie sich Statistiken für CloudWatch Metriken und bereinigen Sie dann Ressourcen.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
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();
    }
}
```

Definieren Sie Funktionen, die vom Szenario aufgerufen werden, um Startvorlagen und Metriken zu verwalten. Diese Funktionen umfassen Auto Scaling EC2, Amazon und CloudWatch Aktionen.

```
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;
    }

}
```

- API-Details finden Sie in den folgenden Themen der AWS SDK für .NET -API-Referenz.
  - CreateAutoScalingGroup
  - DeleteAutoScalingGroup
  - DescribeAutoScalingGroups
  - DescribeAutoScalingInstances
  - DescribeScalingActivities
  - DisableMetricsCollection
  - EnableMetricsCollection
  - SetDesiredCapacity
  - TerminateInstanceInAutoScalingGroup
  - UpdateAutoScalingGroup

Aktionen

`CreateAutoScalingGroup`

Das folgende Codebeispiel zeigt die Verwendung`CreateAutoScalingGroup`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
/// <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;
        }
    }
```

- Einzelheiten zur API finden Sie CreateAutoScalingGroupin der AWS SDK für .NET API-Referenz.

`DescribeAutoScalingGroups`

Das folgende Codebeispiel zeigt die Verwendung`DescribeAutoScalingGroups`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <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;
    }
```

- Einzelheiten zur API finden Sie DescribeAutoScalingGroupsin der AWS SDK für .NET API-Referenz.

`DescribeAutoScalingInstances`

Das folgende Codebeispiel zeigt die Verwendung`DescribeAutoScalingInstances`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <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;
    }
```

- Einzelheiten zur API finden Sie DescribeAutoScalingInstancesin der AWS SDK für .NET API-Referenz.

`DescribeScalingActivities`

Das folgende Codebeispiel zeigt die Verwendung`DescribeScalingActivities`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <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;
    }
```

- Einzelheiten zur API finden Sie DescribeScalingActivitiesin der AWS SDK für .NET API-Referenz.

`DisableMetricsCollection`

Das folgende Codebeispiel zeigt die Verwendung`DisableMetricsCollection`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <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;
    }
```

- Einzelheiten zur API finden Sie DisableMetricsCollectionin der AWS SDK für .NET API-Referenz.

`EnableMetricsCollection`

Das folgende Codebeispiel zeigt die Verwendung`EnableMetricsCollection`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <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;
    }
```

- Einzelheiten zur API finden Sie EnableMetricsCollectionin der AWS SDK für .NET API-Referenz.

`SetDesiredCapacity`

Das folgende Codebeispiel zeigt die Verwendung`SetDesiredCapacity`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <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;
    }
```

- Einzelheiten zur API finden Sie SetDesiredCapacityin der AWS SDK für .NET API-Referenz.

`TerminateInstanceInAutoScalingGroup`

Das folgende Codebeispiel zeigt die Verwendung`TerminateInstanceInAutoScalingGroup`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <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;
    }
```

- Einzelheiten zur API finden Sie TerminateInstanceInAutoScalingGroupin der AWS SDK für .NET API-Referenz.

`UpdateAutoScalingGroup`

Das folgende Codebeispiel zeigt die Verwendung`UpdateAutoScalingGroup`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <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;
        }
    }
```

- Einzelheiten zur API finden Sie UpdateAutoScalingGroupin der AWS SDK für .NET API-Referenz.

Amazon Bedrock — Beispiele für die Verwendung von SDK für .NET (v4)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK für .NET (v4) mit Amazon Bedrock Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zur Einrichtung und Ausführung des Codes im Kontext finden.

Erste Schritte

Hallo Amazon Bedrock

Die folgenden Codebeispiele zeigen, wie Sie mit Amazon Bedrock beginnen können.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
using 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)}");
    }
}
```

- Einzelheiten zur API finden Sie ListFoundationModelsin der AWS SDK für .NET API-Referenz.

Themen

- Aktionen

Aktionen

`ListFoundationModels`

Das folgende Codebeispiel zeigt die Verwendung`ListFoundationModels`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Listet die verfügbaren Bedrock Foundation-Modelle auf.

```
   /// <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);
       }
   }
```

- Einzelheiten zur API finden Sie ListFoundationModelsin der AWS SDK für .NET API-Referenz.

Amazon Bedrock Runtime-Beispiele mit SDK für .NET (v4)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK für .NET (v4) mit Amazon Bedrock Runtime Aktionen ausführen und allgemeine Szenarien implementieren.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zur Einrichtung und Ausführung des Codes im Kontext finden.

Themen

- Anthropic Claude
- Cohere Command
- Meta-Lama
- Mistral KI

Anthropic Claude

Converse

Das folgende Codebeispiel zeigt, wie Sie mithilfe der Converse-API von Bedrock eine Textnachricht an Anthropic Claude senden.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Senden Sie mithilfe der Converse-API von Bedrock eine Textnachricht an Anthropic Claude.

```
// 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;
```

```
}
```

- Einzelheiten zur API finden Sie unter Converse in der API-Referenz.AWS SDK für .NET

ConverseStream

Das folgende Codebeispiel zeigt, wie Sie mithilfe der Converse-API von Bedrock eine Textnachricht an Anthropic Claude senden und den Antwortstream in Echtzeit verarbeiten.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Senden Sie mithilfe der Converse-API von Bedrock eine Textnachricht an Anthropic Claude und verarbeiten Sie den Antwortstream in Echtzeit.

```
// 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;
}
```

- Einzelheiten zur API finden Sie ConverseStreamin AWS SDK für .NET der API-Referenz.

InvokeModel

Das folgende Codebeispiel zeigt, wie mithilfe der Invoke Model API eine Textnachricht an Anthropic Claude gesendet wird.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Verwenden Sie die Invoke Model API, um eine Textnachricht zu senden.

```
// 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;
}
```

- Einzelheiten zur API finden Sie InvokeModelunter AWS SDK für .NET API-Referenz.

Cohere Command

Converse

Das folgende Codebeispiel zeigt, wie mithilfe der Converse-API von Bedrock eine Textnachricht an Cohere Command gesendet wird.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Senden Sie mithilfe der Converse-API von Bedrock eine Textnachricht an Cohere Command.

```
// Use the 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;
}
```

- Einzelheiten zur API finden Sie unter Converse in der API-Referenz.AWS SDK für .NET

ConverseStream

Das folgende Codebeispiel zeigt, wie Sie mithilfe der Converse-API von Bedrock eine Textnachricht an Cohere Command senden und den Antwortstream in Echtzeit verarbeiten.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Senden Sie mithilfe der Converse-API von Bedrock eine Textnachricht an Cohere Command und verarbeiten Sie den Antwortstream in Echtzeit.

```
// Use the 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;
}
```

- Einzelheiten zur API finden Sie ConverseStreamin AWS SDK für .NET der API-Referenz.

Meta-Lama

Converse

Das folgende Codebeispiel zeigt, wie Sie mithilfe der Converse-API von Bedrock eine Textnachricht an Meta Llama senden.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Senden Sie mithilfe der Converse-API von Bedrock eine Textnachricht an Meta Llama.

```
// Use the 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;
}
```

- Einzelheiten zur API finden Sie unter Converse in der API-Referenz.AWS SDK für .NET

ConverseStream

Das folgende Codebeispiel zeigt, wie Sie mithilfe der Converse-API von Bedrock eine Textnachricht an Meta Llama senden und den Antwortstream in Echtzeit verarbeiten.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Senden Sie mithilfe der Converse-API von Bedrock eine Textnachricht an Meta Llama und verarbeiten Sie den Antwortstream in Echtzeit.

```
// Use the 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;
}
```

- Einzelheiten zur API finden Sie ConverseStreamin AWS SDK für .NET der API-Referenz.

Mistral KI

Converse

Das folgende Codebeispiel zeigt, wie Sie mithilfe der Converse-API von Bedrock eine Textnachricht an Mistral senden.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Senden Sie mithilfe der Converse-API von Bedrock eine Textnachricht an Mistral.

```
// Use the 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;
}
```

- Einzelheiten zur API finden Sie unter Converse in der API-Referenz.AWS SDK für .NET

ConverseStream

Das folgende Codebeispiel zeigt, wie Sie mithilfe der Converse-API von Bedrock eine Textnachricht an Mistral senden und den Antwortstream in Echtzeit verarbeiten.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Senden Sie mithilfe der Converse-API von Bedrock eine Textnachricht an Mistral und verarbeiten Sie den Antwortstream in Echtzeit.

```
// Use the 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;
}
```

- Einzelheiten zur API finden Sie in der API-Referenz. ConverseStreamAWS SDK für .NET

AWS CloudFormation Beispiele mit SDK für .NET (v4)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie AWS SDK für .NET (v4) mit verwenden AWS CloudFormation.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Erste Schritte

Hallo AWS CloudFormation

Das folgende Codebeispiel zeigt, wie Sie mit der Verwendung beginnen AWS CloudFormation.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
using 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;
        }
    }
```

- Einzelheiten zur API finden Sie DescribeStackResourcesin der AWS SDK für .NET API-Referenz.

CloudWatch Beispiele mit SDK für .NET (v4)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie AWS SDK für .NET (v4) mit verwenden CloudWatch.

Bei Grundlagen handelt es sich um Code-Beispiele, die Ihnen zeigen, wie Sie die wesentlichen Vorgänge innerhalb eines Services ausführen.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Erste Schritte

Hallo CloudWatch

Die folgenden Codebeispiele veranschaulichen, wie Sie mit der Verwendung von CloudWatch beginnen.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
using 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();
            }
        }
    }
}
```

- Einzelheiten zur API finden Sie ListMetricsin der AWS SDK für .NET API-Referenz.

Themen

- Grundlagen
- Aktionen

Grundlagen

Erlernen der Grundlagen

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Listet CloudWatch Namespaces und Metriken auf.
- Rufen Sie Statistiken für eine Metrik und die geschätzte Fakturierung ab.
- Erstellen und aktualisieren Sie ein Dashboard.
- Erstellen Sie eine Metrik und fügen Sie ihr Daten hinzu.
- Erstellen und lösen Sie einen Alarm aus und zeigen Sie dann den Alarmverlauf an.
- Fügen Sie einen Anomaliedetektor hinzu.
- Ermitteln Sie ein Metrik-Image, dann bereinigen Sie die Ressourcen.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Führen Sie ein interaktives Szenario an einem Prompt aus.

```
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;
    }
}
```

Wrapper-Methoden, die vom Szenario für CloudWatch Aktionen verwendet werden.

```
/// <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;
    }
}
```

- API-Details finden Sie in den folgenden Themen der AWS SDK für .NET -API-Referenz.
  - DeleteAlarms
  - DeleteAnomalyDetector
  - DeleteDashboards
  - DescribeAlarmHistory
  - DescribeAlarms
  - DescribeAlarmsForMetric
  - DescribeAnomalyDetectors
  - GetMetricData
  - GetMetricStatistics
  - GetMetricWidgetImage
  - ListMetrics
  - PutAnomalyDetector

- PutDashboard
- PutMetricAlarm
- PutMetricData

Aktionen

`DeleteAlarms`

Das folgende Codebeispiel zeigt, wie man es benutzt`DeleteAlarms`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
/// <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;
}
```

- Einzelheiten zur API finden Sie DeleteAlarmsin der AWS SDK für .NET API-Referenz.

`DeleteAnomalyDetector`

Das folgende Codebeispiel zeigt die Verwendung`DeleteAnomalyDetector`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
   /// <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;
   }
```

- Einzelheiten zur API finden Sie DeleteAnomalyDetectorin der AWS SDK für .NET API-Referenz.

`DeleteDashboards`

Das folgende Codebeispiel zeigt die Verwendung`DeleteDashboards`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <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;
    }
```

- Einzelheiten zur API finden Sie DeleteDashboardsin der AWS SDK für .NET API-Referenz.

`DescribeAlarmHistory`

Das folgende Codebeispiel zeigt die Verwendung`DescribeAlarmHistory`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <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;
    }
```

- Einzelheiten zur API finden Sie DescribeAlarmHistoryin der AWS SDK für .NET API-Referenz.

`DescribeAlarms`

Das folgende Codebeispiel zeigt die Verwendung`DescribeAlarms`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <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;
    }
```

- Einzelheiten zur API finden Sie DescribeAlarmsin der AWS SDK für .NET API-Referenz.

`DescribeAlarmsForMetric`

Das folgende Codebeispiel zeigt die Verwendung`DescribeAlarmsForMetric`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <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>();
    }
```

- Einzelheiten zur API finden Sie DescribeAlarmsForMetricin der AWS SDK für .NET API-Referenz.

`DescribeAnomalyDetectors`

Das folgende Codebeispiel zeigt die Verwendung`DescribeAnomalyDetectors`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <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;
    }
```

- Einzelheiten zur API finden Sie DescribeAnomalyDetectorsin der AWS SDK für .NET API-Referenz.

`DisableAlarmActions`

Das folgende Codebeispiel zeigt die Verwendung`DisableAlarmActions`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <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;
    }
```

- Einzelheiten zur API finden Sie DisableAlarmActionsin der AWS SDK für .NET API-Referenz.

`EnableAlarmActions`

Das folgende Codebeispiel zeigt die Verwendung`EnableAlarmActions`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <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;
    }
```

- Einzelheiten zur API finden Sie EnableAlarmActionsin der AWS SDK für .NET API-Referenz.

`GetDashboard`

Das folgende Codebeispiel zeigt die Verwendung`GetDashboard`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
/// <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;
}
```

- Einzelheiten zur API finden Sie GetDashboardin der AWS SDK für .NET API-Referenz.

`GetMetricData`

Das folgende Codebeispiel zeigt die Verwendung`GetMetricData`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <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;
    }
```

- Einzelheiten zur API finden Sie GetMetricDatain der AWS SDK für .NET API-Referenz.

`GetMetricStatistics`

Das folgende Codebeispiel zeigt die Verwendung`GetMetricStatistics`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <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>();
    }
```

- Einzelheiten zur API finden Sie GetMetricStatisticsin der AWS SDK für .NET API-Referenz.

`GetMetricWidgetImage`

Das folgende Codebeispiel zeigt die Verwendung`GetMetricWidgetImage`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <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;
    }
```

- Einzelheiten zur API finden Sie GetMetricWidgetImagein der AWS SDK für .NET API-Referenz.

`ListDashboards`

Das folgende Codebeispiel zeigt die Verwendung`ListDashboards`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <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;
    }
```

- Einzelheiten zur API finden Sie ListDashboardsin der AWS SDK für .NET API-Referenz.

`ListMetrics`

Das folgende Codebeispiel zeigt die Verwendung`ListMetrics`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <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;
    }
```

- Einzelheiten zur API finden Sie ListMetricsin der AWS SDK für .NET API-Referenz.

`PutAnomalyDetector`

Das folgende Codebeispiel zeigt die Verwendung`PutAnomalyDetector`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <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;
    }
```

- Einzelheiten zur API finden Sie PutAnomalyDetectorin der AWS SDK für .NET API-Referenz.

`PutDashboard`

Das folgende Codebeispiel zeigt die Verwendung`PutDashboard`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
/// <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>();
    }
```

- Einzelheiten zur API finden Sie PutDashboardin der AWS SDK für .NET API-Referenz.

`PutMetricAlarm`

Das folgende Codebeispiel zeigt die Verwendung`PutMetricAlarm`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <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;
    }
```

- Einzelheiten zur API finden Sie PutMetricAlarmin der AWS SDK für .NET API-Referenz.

`PutMetricData`

Das folgende Codebeispiel zeigt die VerwendungPutMetricData.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <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;
    }
```

- Einzelheiten zur API finden Sie PutMetricDatain der AWS SDK für .NET API-Referenz.

Beispiele für Amazon Cognito Identity Provider mit SDK für .NET (v4)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie den AWS SDK für .NET (v4) mit Amazon Cognito Identity Provider verwenden.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zur Einrichtung und Ausführung des Codes im Kontext finden.

Themen

- Aktionen

Aktionen

`ListUserPools`

Das folgende Codebeispiel zeigt die Verwendung`ListUserPools`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <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;
```

```
    }
```

- Einzelheiten zur API finden Sie ListUserPoolsin der AWS SDK für .NET API-Referenz.

AWS Control Tower Beispiele mit SDK für .NET (v4)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie AWS SDK für .NET (v4) mit verwenden AWS Control Tower.

Bei Grundlagen handelt es sich um Code-Beispiele, die Ihnen zeigen, wie Sie die wesentlichen Vorgänge innerhalb eines Services ausführen.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Erste Schritte

Hallo AWS Control Tower

Die folgenden Codebeispiele veranschaulichen, wie Sie mit der Verwendung von AWS Control Tower beginnen.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
using 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}");
        }
    }
}
```

- Einzelheiten zur API finden Sie ListBaselinesin der AWS SDK für .NET API-Referenz.

Themen

- Grundlagen
- Aktionen

Grundlagen

Erlernen der Grundlagen

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Landezonen auflisten.

- Baselines auflisten, aktivieren, abrufen, zurücksetzen und deaktivieren.
- Steuerelemente auflisten, aktivieren, abrufen und deaktivieren.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Führen Sie ein interaktives Szenario durch, in dem AWS Control Tower Funktionen demonstriert werden.

```
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, "4.0", 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;
                                    break;
                                }
                            }

                            alreadyEnabled = true;
```

```
                            Console.WriteLine("No change, the selected baseline was
already enabled.");
                        }

                        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, "4.0",
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;
    }
}
```

Wrapper-Methoden, die vom Szenario aufgerufen werden, um Aurora-Aktionen zu verwalten.

```
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 if already enabled.</returns>
```

```
   public async Task<string?> EnableBaselineAsync(string targetIdentifier, string
baselineIdentifier, string baselineVersion, string identityCenterBaseline)
   {
       try
       {
           var parameters = new List<EnabledBaselineParameter>
           {
               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) when (ex.Message.Contains("already enabled"))
       {
           Console.WriteLine("Baseline is already enabled for this target");
           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;
        }
    }

}
```

- API-Details finden Sie in den folgenden Themen der AWS SDK für .NET -API-Referenz.
  - CreateLandingZone
  - DeleteLandingZone
  - DisableBaseline
  - DisableControl
  - EnableBaseline
  - EnableControl
  - GetControlOperation
  - GetLandingZoneOperation
  - ListBaselines
  - ListEnabledBaselines
  - ListEnabledControls
  - ListLandingZones
  - ResetEnabledBaseline

Aktionen

`DisableBaseline`

Das folgende Codebeispiel zeigt die Verwendung`DisableBaseline`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
   /// <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;
        }
    }
```

- Einzelheiten zur API finden Sie DisableBaselinein der AWS SDK für .NET API-Referenz.

`DisableControl`

Das folgende Codebeispiel zeigt die Verwendung`DisableControl`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <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;
        }
    }
```

- Einzelheiten zur API finden Sie DisableControlin der AWS SDK für .NET API-Referenz.

`EnableBaseline`

Das folgende Codebeispiel zeigt die Verwendung`EnableBaseline`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <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 if already enabled.</returns>
    public async Task<string?> EnableBaselineAsync(string targetIdentifier, string
 baselineIdentifier, string baselineVersion, string identityCenterBaseline)
    {
        try
        {
            var parameters = new List<EnabledBaselineParameter>
            {
                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) when (ex.Message.Contains("already enabled"))
        {
            Console.WriteLine("Baseline is already enabled for this target");
            return null;
        }
        catch (AmazonControlTowerException ex)
        {
            Console.WriteLine($"Couldn't enable baseline. Here's why:
{ex.ErrorCode}: {ex.Message}");
            throw;
        }
    }
```

- Einzelheiten zur API finden Sie EnableBaselinein der AWS SDK für .NET API-Referenz.

`EnableControl`

Das folgende Codebeispiel zeigt die Verwendung`EnableControl`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
   /// <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;
        }
    }
```

- Einzelheiten zur API finden Sie EnableControlin der AWS SDK für .NET API-Referenz.

`GetBaselineOperation`

Das folgende Codebeispiel zeigt die Verwendung`GetBaselineOperation`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
   /// <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;
       }
   }
```

- Einzelheiten zur API finden Sie GetBaselineOperationin der AWS SDK für .NET API-Referenz.

`GetControlOperation`

Das folgende Codebeispiel zeigt die Verwendung`GetControlOperation`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
   /// <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;
       }
   }
```

- Einzelheiten zur API finden Sie GetControlOperationin der AWS SDK für .NET API-Referenz.

`ListBaselines`

Das folgende Codebeispiel zeigt die Verwendung`ListBaselines`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <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;
        }
    }
```

- Einzelheiten zur API finden Sie ListBaselinesin der AWS SDK für .NET API-Referenz.

`ListEnabledBaselines`

Das folgende Codebeispiel zeigt die Verwendung`ListEnabledBaselines`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <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;
```

```
        }
    }
```

- Einzelheiten zur API finden Sie ListEnabledBaselinesin der AWS SDK für .NET API-Referenz.

`ListEnabledControls`

Das folgende Codebeispiel zeigt die Verwendung`ListEnabledControls`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <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;
        }
    }
```

- Einzelheiten zur API finden Sie ListEnabledControlsin der AWS SDK für .NET API-Referenz.

`ListLandingZones`

Das folgende Codebeispiel zeigt die Verwendung`ListLandingZones`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <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;
        }
    }
```

- Einzelheiten zur API finden Sie ListLandingZonesin der AWS SDK für .NET API-Referenz.

`ResetEnabledBaseline`

Das folgende Codebeispiel zeigt die Verwendung`ResetEnabledBaseline`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <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;
       }
```

```
    }
```

- Einzelheiten zur API finden Sie ResetEnabledBaselinein der AWS SDK für .NET API-Referenz.

DynamoDB-Beispiele mit SDK für .NET (v4)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie Aktionen ausführen und allgemeine Szenarien implementieren, indem Sie AWS SDK für .NET (v4) mit DynamoDB verwenden.

Bei Grundlagen handelt es sich um Code-Beispiele, die Ihnen zeigen, wie Sie die wesentlichen Vorgänge innerhalb eines Services ausführen.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Erste Schritte

Hallo DynamoDB

Die folgenden Codebeispiele veranschaulichen die ersten Schritte mit DynamoDB.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
using 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}");
        }
    }
}
```

- Einzelheiten zur API finden Sie ListTablesin der AWS SDK für .NET API-Referenz.

Themen

- Grundlagen
- Aktionen

Grundlagen

Erlernen der Grundlagen

Wie das aussehen kann, sehen Sie am nachfolgenden Beispielcode:

- Erstellen einer Tabelle, die Filmdaten enthalten kann.
- Einfügen, Abrufen und Aktualisieren eines einzelnen Films in der Tabelle.
- Schreiben von Filmdaten in die Tabelle anhand einer JSON-Beispieldatei.
- Abfragen nach Filmen, die in einem bestimmten Jahr veröffentlicht wurden.
- Scan nach Filmen, die in mehreren Jahren veröffentlicht wurden.
- Löschen eines Films aus der Tabelle und anschließendes Löschen der Tabelle.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
/// <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();
        }
    }
}
```

Erstellt eine Tabelle, die Filmdaten enthält.

```
    /// <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;
        }
    }
```

Fügt der Tabelle einen einzelnen Film hinzu.

```

    /// <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;
        }
    }
```

Aktualisiert ein einzelnes Element in einer Tabelle.

```
    /// <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;
        }
    }
```

Ruft ein einzelnes Element aus der Filmtabelle ab.

```
    /// <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;
        }
    }
```

Schreibt einen Stapel von Elementen in die Filmtabelle.

```
    /// <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;
        }
    }
```

Löscht ein einzelnes Element aus der Tabelle.

```
    /// <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;
        }
    }
```

Fragt die Tabelle nach Filmen ab, die in einem bestimmten Jahr veröffentlicht wurden.

```
/// <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;
        }
    }
```

Scannt die Tabelle nach Filmen, die in einem bestimmten Zeitraum veröffentlicht wurden.

```
    /// <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;
        }
    }
```

Löscht die Filmtabelle.

```
    /// <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;
        }
    }
```

- API-Details finden Sie in den folgenden Themen der AWS SDK für .NET -API-Referenz.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable
  - DescribeTable
  - GetItem
  - PutItem
  - Abfrage
  - Scan
  - UpdateItem

Aktionen

`BatchWriteItem`

Das folgende Codebeispiel zeigt die Verwendung`BatchWriteItem`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Schreibt einen Stapel von Elementen in die Filmtabelle.

```
/// <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;
        }
    }
```

- Einzelheiten zur API finden Sie BatchWriteItemin der AWS SDK für .NET API-Referenz.

`CreateTable`

Das folgende Codebeispiel zeigt die Verwendung`CreateTable`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <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;
        }
    }
```

- Einzelheiten zur API finden Sie CreateTablein der AWS SDK für .NET API-Referenz.

`DeleteItem`

Das folgende Codebeispiel zeigt die Verwendung`DeleteItem`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <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;
        }
    }
```

- Einzelheiten zur API finden Sie DeleteItemin der AWS SDK für .NET API-Referenz.

`DeleteTable`

Das folgende Codebeispiel zeigt die Verwendung`DeleteTable`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <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;
        }
    }
```

- Einzelheiten zur API finden Sie DeleteTablein der AWS SDK für .NET API-Referenz.

`GetItem`

Das folgende Codebeispiel zeigt die Verwendung`GetItem`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <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;
        }
    }
```

- Einzelheiten zur API finden Sie GetItemin der AWS SDK für .NET API-Referenz.

`PutItem`

Das folgende Codebeispiel zeigt die Verwendung`PutItem`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <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;
        }
    }
```

- Einzelheiten zur API finden Sie PutItemin der AWS SDK für .NET API-Referenz.

`Query`

Das folgende Codebeispiel zeigt die Verwendung`Query`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
/// <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;
        }
    }
```

- Weitere API-Informationen finden Sie unter Query in der AWS SDK für .NET -API-Referenz.

Scan

Das folgende Codebeispiel zeigt die VerwendungScan.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <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;
        }
    }
```

- Weitere API-Informationen finden Sie unter Scan in der AWS SDK für .NET -API-Referenz.

`UpdateItem`

Das folgende Codebeispiel zeigt die Verwendung`UpdateItem`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
    /// <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;
        }
    }
```

- Einzelheiten zur API finden Sie UpdateItemin der AWS SDK für .NET API-Referenz.

EC2 Amazon-Beispiele mit SDK für .NET (v4)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK für .NET (v4) mit Amazon Aktionen ausführen und allgemeine Szenarien implementieren EC2.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zum Einrichten und Ausführen des Codes im Kontext finden.

Erste Schritte

Hallo Amazon EC2

Die folgenden Codebeispiele zeigen, wie Sie mit Amazon beginnen können EC2.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

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

- Einzelheiten zur API finden Sie DescribeSecurityGroupsin der AWS SDK für .NET API-Referenz.

Amazon ECS-Beispiele mit SDK für .NET (v4)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK für .NET (v4) mit Amazon ECS Aktionen ausführen und allgemeine Szenarien implementieren.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zur Einrichtung und Ausführung des Codes im Kontext finden.

Erste Schritte

Hallo Amazon ECS

Das folgende Codebeispiel zeigt die ersten Schritte mit Amazon ECS.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

```
using 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.");
```

```
        }

    }
}
```

- Einzelheiten zur API finden Sie ListClustersin der AWS SDK für .NET API-Referenz.

Amazon S3 S3-Beispiele mit SDK für .NET (v4)

Die folgenden Codebeispiele zeigen Ihnen, wie Sie mithilfe von AWS SDK für .NET (v4) mit Amazon S3 Aktionen ausführen und allgemeine Szenarien implementieren.

Aktionen sind Codeauszüge aus größeren Programmen und müssen im Kontext ausgeführt werden. Während Aktionen Ihnen zeigen, wie Sie einzelne Service-Funktionen aufrufen, können Sie Aktionen im Kontext der zugehörigen Szenarios anzeigen.

Szenarien sind Code-Beispiele, die Ihnen zeigen, wie Sie bestimmte Aufgaben ausführen, indem Sie mehrere Funktionen innerhalb eines Services aufrufen oder mit anderen AWS-Services kombinieren.

Jedes Beispiel enthält einen Link zum vollständigen Quellcode, in dem Sie Anweisungen zur Einrichtung und Ausführung des Codes im Kontext finden.

Themen

- Aktionen
- Szenarien

Aktionen

`CreatePresignedPost`

Das folgende Codebeispiel zeigt die Verwendung`CreatePresignedPost`.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Erstellen Sie eine vorsignierte POST-URL.

```
    /// <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);
    }
```

- Einzelheiten zur API finden Sie CreatePresignedPostin der AWS SDK für .NET API-Referenz.

Szenarien

Eine vorsignierte URL erstellen

Das folgende Codebeispiel zeigt, wie Sie eine vorsignierte URL für Amazon S3 erstellen und ein Objekt hochladen.

SDK für .NET (v4)

> **Note**
>
> Es gibt noch mehr dazu GitHub. Hier finden Sie das vollständige Beispiel und erfahren, wie Sie das AWS -Code-Beispiel- einrichten und ausführen.

Erstellen und verwenden Sie vorsignierten POST URLs für direkte Browser-Uploads.

```
/// <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");
            }
        }
    }
}
```

Migrieren Sie Ihr Projekt für die AWS SDK für .NET

Dieser Abschnitt enthält Informationen zu Migrationsaufgaben für Version 4 von und Anweisungen zur Ausführung dieser Aufgaben. AWS SDK für .NET Informationen zur Migration zu früheren Versionen (z. B. zur Migration von V2 zu V3 oder von einer V3-Subversion zu einer anderen) finden Sie unter Migrieren Ihres Projekts im AWS SDK für .NET Entwicklerhandbuch (Version 3).

Weitere Informationen finden Sie auch unter Was ist neu und Unterstützte Plattformen.

Themen

- Migration auf Version 4 von AWS SDK für .NET
- Migration von .NET-Standard 1.3

Migration auf Version 4 von AWS SDK für .NET

Die AWS SDK für .NET Version 4 (V4) enthält eine beträchtliche Anzahl wichtiger Änderungen gegenüber Version 3 (V3) des SDK. In diesem Thema werden die grundlegenden Änderungen in Version 4 und mögliche Arbeiten beschrieben, die Sie möglicherweise durchführen müssen, um Ihre Umgebung oder Ihren Code von Version 3 zu migrieren. Weitere Informationen zu anderen wichtigen Änderungen im SDK finden Sie in den folgenden Ressourcen:

- Das Development-Tracker-Problem am:. GitHub https://github.com/aws/aws-sdk-net/issues/3362
- Der Blogbeitrag Vorschau 1 von V4. AWS SDK für .NET
- Der Blogbeitrag Vorschau 4 von AWS SDK für .NET V4.
- Der Blogbeitrag Allgemeine Verfügbarkeit von AWS SDK für .NET V4.0.

.NET Framework.

Das.NET Framework 3.5-Ziel wurde aus Version 4 von entfernt. AWS SDK für .NET Infolgedessen unterstützt das SDK das.NET Framework 3.5 nicht mehr. Diese Version des SDK wurde für .NET Framework 4.7.2 kompiliert und läuft in der.NET 4.0-Laufzeit. Weitere Informationen finden Sie unter Unterstützte Plattformen.

Wertetypen

Eigenschaften, die Werttypen in Klassen verwenden, die für Anfragen und Antworten verwendet werden, wurden dahingehend geändert, dass sie Werttypen verwenden, die NULL-Werte zulassen. Eigenschaften mit den folgenden Typen wurden geändert:

- `bool`wurde geändert in `bool?`
- `double`wurde geändert zu `double?`
- `int`wurde geändert zu `int?`
- `float`wurde geändert zu `float?`
- `long`wurde geändert zu `long?`
- `Datetime`wurde geändert zu `Datetime?`

Weitere Informationen zu dieser Änderung finden Sie im Blogbeitrag Preview 1 of AWS SDK für .NET V4.

Sammlungen

Eigenschaften, die Sammlungen in Klassen verwenden, die für Anfragen und Antworten verwendet werden, verwenden jetzt standardmäßig den Wert`null`. Daher muss Ihr Code überprüfen, ob eine Sammlung nicht Null ist, bevor Sie versuchen, sie zu verwenden. Zum Beispiel:

```
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.
    }
}
```

Das V3-Verhalten beim Initialisieren von Sammlungen kann wiederhergestellt werden, indem Sie `Amazon.AWSConfigs.InitializeCollections` auf `true` setzen. Diese Eigenschaft gibt es auch in V3 für Benutzer, die diese Verhaltensänderung ausprobieren möchten, bevor sie auf V4 aktualisieren.

Weitere Informationen zu dieser Änderung finden Sie im Blogbeitrag Preview 1 of AWS SDK für .NET V4.

AWS Security Token Service (STS)

- Der regionale Endpunkt

  Wenn Sie Anbieter für Anmeldeinformationen verwenden, die darauf angewiesen sind AWS STS, verwenden die Anrufe immer den regionalen Endpunkt. Dies unterscheidet sich von Version 3 des SDK, bei der standardmäßig die `us-east-1` Region verwendet wurde, wenn sie in der öffentlichen Partition ausgeführt wurde, unabhängig von der konfigurierten Region.

- Die **`StsRegionalEndpointsValue`** Aufzählung

  Die `StsRegionalEndpointsValue` Aufzählung wurde aus dem Amazon.Runtime-Namespace entfernt. Jeder Code, der diese Aufzählung verwendet, sollte entfernt werden.

- Die **`STSAssumeRoleAWSCredentials`**-Klasse

  Der veraltete STS-Anbieter für Anmeldeinformationen zur Übernahme der RolleSTSAssumeRoleAWSCredentials, wurde aus Amazon entfernt. SecurityTokenNamespace. Verwenden Sie stattdessen AssumeRoleAWSCredentialsvon Amazon.Runtime.

Änderungen im Zusammenhang mit **`ClientConfig`**

Die Amazon.Runtime. ClientConfigKlasse ist die Basisklasse von Service-Client-Konfigurationsklassen wie AmazonS3Config. Die folgenden Änderungen wurden an dieser Basisklasse vorgenommen.

- Standardmäßiger Wiederholungsmodus

  Die `RetryMode` Eigenschaft ist standardmäßig auf `Standard` statt. `Legacy` Aus diesem Grund wurde der `Legacy` Wert aus der Amazon.Runtime entfernt. RequestRetryModeAufzählung.

- Standardkonfigurationsmodus

  Die `DefaultConfigurationMode` Eigenschaft ist standardmäßig auf `Standard` statt. `Legacy` Aus diesem Grund wurde der `Legacy` Wert aus der Amazon.Runtime entfernt. DefaultConfigurationModeAufzählung.

- Das Eigentum **`ReadWriteTimeout`**

Die veraltete `ReadWriteTimeout` Eigenschaft wurde aus allen Zielen außer .NET Framework 4.7.2 entfernt.

Die AWSSDK .Extensions. NETCore.Setup-Paket NuGet

Die AWSSDK.Extensions. NETCore NuGet .Setup-Paket wurde aktualisiert, um Probleme zu beheben, die in Version 3 des SDK auftraten, und um das Paket für Native AOT sicher zu machen. Diese Änderungen sind im Folgenden zusammengefasst. Ausführliche Informationen finden Sie unter PR 3353 im aws-sdk-netRepository unter GitHub.

- Die **`DefaultClientConfig`**-Klasse

  Die `DefaultClientConfig` Klasse wird nicht mehr von der Basisklasse Amazon.Runtime für die Service-Client-Konfiguration übernommen. ClientConfig. Die relevanten Eigenschaften von `ClientConfig` wurden `DefaultClientConfig` unter Verwendung von Nullwerttypen repliziert. Diese Änderung ermöglicht es uns zu erkennen, wann ein Wert aktiviert wurde, `DefaultClientConfig` wenn die Werte in die Konfiguration kopiert werden, die für den Service-Client erstellt wird.

  Ein besonderes Ergebnis dieser Änderung `DefaultClientConfig.HttpClientFactory` ist, dass sie in Version 4 nicht mehr verfügbar ist. Verwenden Sie stattdessen `AWSConfigs.HttpClientFactory`. Weitere Informationen finden Sie in GitHub Ausgabe 3790.

  Ein weiteres Ergebnis dieser Änderung ist, dass die generischen Überladungen der `IConfiguration.GetAWSOptions` Erweiterungsmethode, die ein Dienstkonfigurationsobjekt akzeptiert hat, entfernt wurden. Stattdessen sollte die nicht generische Überladung verwendet werden, und das SDK kümmert sich automatisch um das Ausfüllen dienstspezifischer Einstellungen. Weitere Informationen finden Sie in Ausgabe 3866. GitHub

- Natives AOT

  Dem Paket wurde ein neuer Mechanismus zum Erstellen von Service-Clients hinzugefügt, der statische C# 11-Schnittstellenmethoden verwendet. Durch diese Änderung entfällt die Notwendigkeit, Assembly-Ladevorgänge durchzuführen, um Instanzen von Dienstclients zu erstellen, einschließlich der Bearbeitung von Zeichenketten des Dienstschnittstellennamens zur Berechnung des Dienstclienttyps, der mit Native AOT nicht kompatibel ist. Diese Änderung ist nur für.NET 8 und höher verfügbar. Ältere Versionen verwenden immer noch den ursprünglichen Mechanismus.

Weitere Informationen zu diesem Paket in diesem Handbuch finden Sie unterAWSSDK.Erweiterungen. NETCore.Setup und IConfiguration. Der Quellcode für dieses Paket befindet sich GitHub unterhttps://github.com/aws/aws-sdk-net/tree/main/extensions/src/AWSSDK.Extensions.NETCore.Setup.

`CookieSigner` und `UrlSigner`

Die `UrlSigner` Erweiterungen `CookieSigner` und Erweiterungen für Amazon CloudFront wurden in ein separates Erweiterungspaket namens AWSSDK.Extensions verschoben. CloudFront.Unterzeichner. Diese Änderung soll OpenSSL 3 unterstützen und eine Abhängigkeit von BouncyCastle.Cryptography übernehmen.

Der Quellcode für dieses Paket befindet sich unter. GitHub https://github.com/aws/aws-sdk-net/tree/main/extensions/src/AWSSDK.Extensions.CloudFront.Signers

DateTime im Vergleich zu UTC DateTime

Einige V3-Klassen verfügen über eine DateTime Eigenschaft, die als „veraltet“ oder „veraltet“ gekennzeichnet ist, sowie über eine alternative DateTime UTC-Eigenschaft. In diesen Klassen wurde die veraltete DateTime Eigenschaft entfernt und der Name der DateTime UTC-Eigenschaft wurde in den ursprünglichen Namen der DateTime Eigenschaft geändert.

Im Folgenden finden Sie einige Beispiele für Klassen, für die diese Änderung implementiert wurde.

- DescribeSpotPriceHistoryRequest:
  - Die veraltete `StartTime` Eigenschaft wurde entfernt und der Name der `StartTimeUtc` Eigenschaft wurde in "StartTime" geändert.
  - Die veraltete `EndTime` Eigenschaft wurde entfernt und der Name der `EndTimeUtc` Eigenschaft wurde in "EndTime" geändert.
- CreateFleetRequest
  - Die veraltete `ValidFrom` Eigenschaft wurde entfernt und der Name der `ValidFromUtc` Eigenschaft wurde in "ValidFrom" geändert.
  - Die veraltete `ValidUntil` Eigenschaft wurde entfernt und der Name der `ValidUntilUtc` Eigenschaft wurde in "ValidUntil" geändert.

Diese Änderung kann zu Verzögerungen führen, wenn eine Anwendung die ursprüngliche, veraltete DateTime Eigenschaft verwendet. Bei Code, der die DateTime UTC-Eigenschaft verwendet, tritt ein Kompilierungsfehler auf.

DateTime Parsen

Die DateTimeUnmarshaller Klasse wurde aktualisiert. Diese Klasse hatte DateTime Zeichenketten analysiert und als Ortszeit zurückgegeben. In einigen Fällen wurden diese Werte aufgrund eines früheren Updates wieder in UTC konvertiert, aber nicht immer. Nun wird davon ausgegangen, dass DateTime Zeichenketten, bei denen kein Marshalling vorgenommen wurde, UTC sind, und sie werden dann als UTC spezifiziert und unmarshallt. Dieses Update beinhaltet die folgenden Verhaltensänderungen.

Bestimmte Zeitstempeleigenschaften, die auf der DateTime Klasse basieren, wurden anhand von lokalen Zeitangaben analysiert. Dazu gehörten Antwort-Unmarshaller für Zeitstempel und Listenzeitstempel für Formate und. `TimestampFormat.ISO8601` `TimestampFormat.RFC822` DateTime Das Parsen wurde aktualisiert und gibt stattdessen UTC-Zeiten zurück.

`ConvertFromUnixEpochSeconds` und **`ConvertFromUnixEpochMilliseconds`**

Die ConvertFromUnixEpochMillisecondsMethoden ConvertFromUnixEpochSecondsund, die Sekunden der Unix-Epoche in eine DateTime Struktur umwandeln, gaben die Unix-Epochenzeit als Ortszeit statt als UTC-Zeit zurück. Diese Methoden geben jetzt die UTC-Zeit zurück.

Protokollierung

Die Art und Weise, wie Sie die Protokollierung im SDK aktivieren, wurde für Version 4 aktualisiert. Die Protokollierung an der Konsole und die Systemdiagnose funktionieren genauso wie V3, d. h. die `LoggingConfig.LogTo` Eigenschaft der AWSConfigsKlasse wird entweder auf `LoggingOptions.Console` oder gesetzt`LoggingOptions.SystemDiagnostics`. Die LoggingOptionsOption für `log4net` wurde zusammen mit der internen Logik des SDK für die Verwendung von Reflection zum Anhängen an eine In-Memory-Instanz von `log4net` entfernt.

Um die SDK-Protokollierung in ein Logging-Framework einzubinden, wird ein separates Adapterpaket verwendet, um das SDK mit dem Logging-Framework zu verbinden. Verwenden Sie das AWSSDK.Extensions.Logging.Log4-Paket für und das NetAdaptor .Extensions.Logging. `log4net` AWSSDK ILogger`Microsoft.Extensions.Logging`Adapterpaket für. Die folgenden Codebeispiele zeigen Ihnen, wie Sie die Protokollierung in diesen beiden Fällen konfigurieren.

Beispielkonfiguration für log4net

Fügen Sie das `AWSSDK.Extensions.Logging.Log4NetAdaptor` NuGet Paket hinzu und rufen Sie die statische `ConfigureAWSSDKLogging` Methode von auf. `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));
```

Beispielkonfiguration für Microsoft.Extensions.Logging

Fügen Sie das `AWSSDK.Extensions.Logging.ILoggerAdaptor` NuGet Paket hinzu und rufen Sie die `ConfigureAWSSDKLogging` Erweiterungsmethode über die Schnittstelle auf. `ILoggerFactory`

```
var builder = WebApplication.CreateBuilder(args);

var app = builder.Build();

app.Services.GetRequiredService<ILoggerFactory>()
    .ConfigureAWSSDKLogging();
```

Support für HTTP 2

Support für HTTP 2 wurde hinzugefügt, um bidirektionales Streaming zu ermöglichen. Weitere Informationen finden Sie unter Support für HTTP 2.

Single Sign-On

Der Standardwert der `SupportsGettingNewToken` Eigenschaft der SSOAWSCredentialsOptions-Klasse wurde von `true` auf `false` geändert. Wenn Sie über Anwendungen verfügen, die die SSOAWSCredentialsKlasse zum Abrufen von SSO-Anmeldeinformationen verwenden, müssen Sie die `Options.SupportsGettingNewToken` Eigenschaft möglicherweise auf setzen`true`. Beispiele für diese Konfiguration finden Sie in den Codebeispielen unterTutorial für SSO mit

ausschließlich.NET-Anwendungen. Weitere Informationen finden Sie unter PR 3737 im aws-sdk-net GitHub Repository.

DynamoDB-spezifische Änderungen

Die folgenden Änderungen sind spezifisch für Amazon DynamoDB. Bei vielen von ihnen handelt es sich um grundlegende Änderungen.

Weitere Informationen zu Änderungen an DynamoDB in Version 4 von finden Sie im Blogbeitrag Preview 4 of AWS SDK für .NET V4. AWS SDK für .NET

Zum Anzeigen von Artikeln öffnen

V4-Änderungen im SDK für DynamoDB beheben einige Probleme im Zusammenhang mit der Testbarkeit, konzentrieren sich jedoch hauptsächlich auf die Bibliotheken auf hoher Ebene:

- Das.NET-Dokumentmodell, das DocumentModelim Code benannt ist.
- Das.NET-Objektpersistenzmodell, das DataModelim Code benannt ist.

Ausführliche Informationen zu diesen Programmiermodi finden Sie DynamoDB in diesem Handbuch.

Dokumentmodell: Die Ausnahme für die gefälschte Schnittstelle wurde **`IAmazonDynamoDB`** aktualisiert

Wenn im Dokumentenmodell vor Version 4 des SDK eine Tabelle mit einer simulierten IAmazon DynamoDB-Schnittstelle initialisiert wurde, wurde sie zurückgegeben. `NullReferenceException` Stattdessen kehrt Version 4 des SDK zurück. `InvalidOperationException` Asynchrone `Table` Methoden sollten mit einem gefälschten Client funktionieren, aber es kann trotzdem zu Ausnahmen kommen, wenn Sie synchrone Methoden von aufrufen. `.NET/Core/Standard`

Weitere Informationen zu dieser Änderung finden Sie unter PR 3388 unter. GitHub

Modell des Dokuments: **`FromJson`** und Methoden **`ToJson`**

Die `ToJson` Methoden `FromJson` und der Document-Klasse werden jetzt `System.Text.Json` anstelle von LitJson Serialisierung verwendet und LitJson wurden aus Version 4 des SDK entfernt. Ein Vorteil der Verwendung `System.Text.Json` besteht darin, dass dieser Parser die Verwendung `Decimal` des.NET-Typs unterstützt, der eine höhere Genauigkeit für numerische Gleitkommaeigenschaften unterstützt.

Object Persistence Model: Die Klasse **`DynamoDBOperationConfig`**

Im Objektpersistenzmodell wurden die folgenden Änderungen an der gemeinsam genutzten Dynamo DBOperation Config-Klasse vorgenommen:

- Die Klasse wurde in neue betriebsspezifische Klassen wie SaveConfig, und aufgeteilt. LoadConfigQueryConfig Methoden, die dies erfordern, `DynamoDBOperationConfig` wurden als veraltet markiert und können in future entfernt werden.

  Weitere Informationen zu dieser Änderung finden Sie in PR 3421 unter GitHub.
- Die `DisableFetchingTableMetadata` Eigenschaften `MetadataCachingMode` und wurden aus der Klasse entfernt. Diese Eigenschaften waren nicht in den zuvor erwähnten neuen betriebsspezifischen Klassen enthalten. Bei den entfernten Eigenschaften handelt es sich um Einstellungen auf Tabellenebene, die für die globale `Context` Eigenschaft der DynamoDB-Klasse oder für die AWSConfigsDynamo Config-Klasse angegeben werden sollten. DBContext

  Weitere Informationen zu dieser Änderung finden Sie in PR 3422 unter. GitHub
- Die Klasse erbt nicht mehr von der Dynamo DBContext Config-Klasse. Dadurch wird verhindert, dass Sie ein `DynamoDBOperationConfig` Objekt an den Konstruktor für Dynamo übergebenDBContext, wo einige Eigenschaften der betriebsspezifischen Konfiguration (z. B.) nicht zutreffen. `OverrideTableName`

  Weitere Informationen zu dieser Änderung finden Sie in PR 3422 unter. GitHub

Modell der Objektpersistenz: Polymorphismus

Die DBPolymorphicTypeAttributeDynamo-Klasse wurde dem Objektpersistenzmodell hinzugefügt. Diese Klasse ermöglicht die Unterstützung für die Serialisierung und Deserialisierung polymorpher Typen. Weitere Informationen finden Sie unter PR 3643 unter. GitHub

Dokumentmodell und Objektpersistenzmodell: Machbare Operationen

Es wurden neue betriebsspezifische Schnittstellen hinzugefügt, mit denen Kunden DynamoDB-Operationen simulieren können. Die Factory-Methoden auf der IDynamoDBContextSchnittstelle wurden aktualisiert und geben nun die neuen Schnittstellen zurück.

Weitere Informationen zu dieser Änderung finden Sie unter PR 3450 unter GitHub.

- Object Persistence-Modell

  - Simulieren Sie `BatchGet` Operationen über die `IMultiTableBatchGet` Schnittstellen `IBatchGet` und.
  - `BatchWrite`Simulieren Sie Operationen über die `IMultiTableBatchWrite` Schnittstellen `IBatchWrite` und.
  - `TransactGet`Simulieren Sie Operationen über die `IMultiTableTransactGet` Schnittstellen `ITransactGet` und.
  - `TransactWrite`Simulieren Sie Operationen über die `IMultiTableTransactWrite` Schnittstellen `ITransactWrite` und.
  - `Query`Simulationen `Scan` und Operationen über die `IAsyncSearch` Schnittstelle.
- Dokument-Modell
  - `Table`Scheinoperationen über die `ITable` Schnittstelle.
  - Mock `Scan` und `Query` Operationen über die `ISearch` Schnittstelle.
  - `TransactWrite`Simulieren Sie Operationen über die `IMultiTableDocumentTransactWrite` Schnittstellen `IDocumentTransactWrite` und.
  - `TransactGet`Simulieren Sie Operationen über die `IMultiTableDocumentTransactGet` Schnittstellen `IDocumentTransactGet` und.
  - `BatchWrite`Simulieren Sie Operationen über die `IMultiTableDocumentBatchWrite` Schnittstellen `IDocumentBatchWrite` und.
  - `BatchGet`Simulieren Sie Operationen über die `IMultiTableDocumentBatchGet` Schnittstellen `IDocumentBatchGet` und.

Dokumentenmodell und Objektpersistenzmodell: Support für natives AOT

Eine Einschränkung von Native AOT ist die Unterstützung verschachtelter .NET-Typen. In einigen Fällen können diese verschachtelten Typen von der Trimmkomponente des.NET-Compilers unbemerkt bleiben. In diesem Fall erhalten Sie möglicherweise eine Ausnahme wie: "“ `System.InvalidOperationException: Type <type> is unsupported, it cannot be instantiated.`

Sie können diese Einschränkung umgehen, indem Sie das `DynamicDependency` irgendwo im Codepfad hinzufügen, das den Trimmer über die Abhängigkeit vom Untertyp informiert. Der Konstruktor des.NET-Typs der obersten Ebene, der gespeichert wird, ist ein wahrscheinlicher Ort. Das folgende Codebeispiel zeigt Ihnen, wie Sie das `DynamicDependency` Attribut verwenden:

```
[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; }
}
```

Dynamo DBStreams

Dynamo DBStreams wurde aus dem AWSSDK.dynamoDB-Paket entfernt und ist in einem eigenen NuGet Paket, AWSSDK.Dynamo DBStreams, verfügbar und hat seinen eigenen Namespace. `Amazon.DynamoDBStreams`

Erlaubt das Entfernen **`TableNamePrefix`** des Werts

Sie können jetzt den Wert der `TableNamePrefix` Eigenschaft in der Dynamo DBContext Config-Klasse auf der Ebene der einzelnen Operationen entfernen. Weitere Informationen zu dieser Änderung finden Sie in PR 3476 unter. GitHub

`RetrieveDateTimeInUtc` Eigenschaft

Für die Dynamo DBContext Config-Klasse wurde der Standardwert der `RetrieveDateTimeInUtc` Eigenschaft auf `true` geändert.

`DynamoDBContextTableNamePrefix` Eigenschaft

Die `DynamoDBContextTableNamePrefix` Eigenschaft wurde aus der AWSConfigsDynamoDB-Klasse entfernt. Benutzer sollten anstelle von anrufen `AWSConfigsDynamoDB.Context.TableNamePrefix`

Spezifische Änderungen für EC2

Die folgenden Änderungen sind spezifisch für Amazon EC2. Die meisten oder alle von ihnen sind bahnbrechende Änderungen.

Zum Ansehen von Artikeln öffnen

`GetDecryptedPassword`

Die `GetDecryptedPassword` Erweiterung für Amazon EC2 wurde in ein separates Erweiterungspaket namens AWSSDK.Extensions verschoben. EC2. DecryptPassword. Diese Änderung soll OpenSSL 3 unterstützen und eine Abhängigkeit von BouncyCastle.Cryptography übernehmen.

Der Quellcode für dieses Paket befindet sich unter. GitHub https://github.com/aws/aws-sdk-net/tree/main/extensions/src/AWSSDK.Extensions.EC2.DecryptPassword

Support für Amazon EC2 IMDSv1

Die Support für Instance Metadata Service Version 1 (IMDSv1) wurde entfernt. Version 4 des SDK verwendet beim Abrufen von Anmeldeinformationen und anderen Metadaten aus dem IMDS immer den Instanz-Metadatendienst Version 2 (IMDSv2). Weitere Informationen zum IMDS finden Sie unter Verwenden des IMDS im EC2 Amazon-Benutzerhandbuch.

Programmierelemente, die geändert oder entfernt wurden

- Der gesamte `Amazon.EC2.Import` Namespace und der gesamte Code wurden entfernt.
- Der gesamte `Amazon.EC2.Util` Namespace und der gesamte Code wurden entfernt, einschließlich der AMI-Dienstprogramme, mit denen nach Windows EC2 AMIs gesucht wurde.
- Die veraltete `IpRanges` Eigenschaft wurde aus der IpPermissionKlasse entfernt. Verwenden Sie stattdessen die `Ipv6Ranges` Eigenschaften `Ipv4Ranges` oder.
- Die folgenden veralteten Felder wurden aus der EC2InstanceMetadataKlasse entfernt: `EC2_METADATA_SVCEC2_METADATA_ROOT`,`EC2_USERDATA_ROOT`,`EC2_DYNAMICDATA_ROOT`, und`EC2_APITOKEN_URL`.

Spezifische Änderungen für S3

Die folgenden Änderungen sind spezifisch für Amazon S3. Bei den meisten oder allen handelt es sich um grundlegende Änderungen.

Zum Ansehen von Artikeln öffnen

AWS-Region us-east-1

Für die `us-east-1` Region konfigurierte Amazon S3 S3-Serviceclients können nicht mehr auf Buckets in anderen Regionen zugreifen. Auf Buckets muss mit S3-Serviceclients zugegriffen werden, die für die Region konfiguriert sind, in der sich der Bucket befindet.

Weitere Informationen zu dieser Änderung finden Sie im Blogbeitrag Preview 4 of AWS SDK für .NET V4.

S3-Verschlüsselungsclient

Der Amazon S3 S3-Verschlüsselungsclient, der im `Amazon.S3.Encryption` Namespace definiert ist, wurde aus dem AWSSDK.S3-Paket entfernt. Dieser Client wurde in ein eigenes Paket namens Amazon.Extensions.S3.Encryption verschoben. Die Dokumentation dazu finden Sie unter. https://aws.github.io/amazon-s3-encryption-client-dotnet/api/Amazon.Extensions.S3.Encryption.html Informationen zur Migration finden Sie unter. the section called “Migration des S3-Verschlüsselungsclients” Weitere Informationen zur S3-Verschlüsselung finden Sie unter Unterstützte Verschlüsselungsalgorithmen im Amazon S3 Encryption Client Developer Guide.

S3-Tagging-Direktive für **`CopyObject`**

Die `TaggingDirective` Eigenschaft wurde als öffentliches Eigentum der CopyObjectRequestKlasse bereitgestellt, das von `AmazonS3Client.CopyObject` Methoden verwendet wird. Diese Eigenschaft entspricht dem Amazon S3 `x-amz-tagging-directive` S3-Parameter, wie er in der CopyObjectAktion definiert ist.

Die Tagging-Direktive ist nicht mehr automatisch auf COPY gesetzt. Wenn ein Entwickler keine Tagging-Direktive spezifiziert, geht das S3-Backend automatisch davon aus, dass es sich um COPY handelt. Wenn ein Entwickler die Eigenschaft jedoch explizit auf Null setzt, wird der Wert überhaupt nicht gesetzt.

Die **`UseArnRegion`** Eigenschaft für die S3-Konfiguration

Die `UseArnRegion` Eigenschaft der Klasse Amazon.S3.AmazonS3Config wurde aktualisiert, sodass die `AWS_S3_USE_ARN_REGION` Umgebungsvariable Vorrang vor der Einstellung in der gemeinsam genutzten Datei hat. `s3_use_arn_region` AWS `config` Weitere Informationen zu diesen Variablen und Einstellungen finden Sie in der Referenz zu den Einstellungen im Referenzhandbuch und im Tools-Referenzhandbuch.AWS SDKs

Führende Schrägstriche für die Methoden **`CopyObject`** und **`CopyPart`**

Führende Schrägstriche werden für Amazon S3 `CopyObject` und Methoden nicht mehr gekürzt. `CopyPart` Die `DisableTrimmingLeadingSlash` Eigenschaft wurde aus den Klassen und entfernt. CopyObjectRequestCopyPartRequest

Die **`DoesS3BucketExist...`** Methoden

Die veralteten `DoesS3BucketExistAsync` Methoden `DoesS3BucketExist` und wurden aus der AmazonS3Util-Klasse entfernt, die die AmazonS3-Schnittstelle implementiert. ICore Diese Methoden wurden entfernt, da sie immer HTTP verwenden. Verwenden Sie stattdessen DoeSS3 BucketExist V2 und DoeSS3 BucketExist V2Async.

SDK verwendet immer SigV4

Version 4 von verwendet AWS SDK für .NET immer AWS Signature Version 4 (Sigv4) zum Signieren von Anfragen. Diese Änderung führt zu den folgenden verwandten Änderungen:

- Die `UseSignatureVersion4` Eigenschaft der Klasse AWSConfigsS3 wurde entfernt.
- Die `SignatureVersion` Eigenschaft von Amazon.Runtime. ClientConfigDie Klasse wurde entfernt. Diese Eigenschaft wurde aus Gründen der Abwärtskompatibilität nur von Amazon S3 verwendet.
- Die `RegionEndpoint.Endpoint` Klasse wurde entfernt. Dazu gehört die `SignatureVersionOverride` Eigenschaft, die verwendet wurde, um Signaturversionen für Amazon S3 zu überschreiben. Verwenden Sie stattdessen die dienstspezifische `client.DetermineServiceOperationEndPoint()` Methode.
- Aktualisierte Methoden AmazonS3Util. PostUploadPostUploadSignedPolicyund S3. GetSignedPolicyum SigV4 zu verwenden. Infolgedessen wurde die `S3PostUploadSignedPolicy.GetSignedPolicyV4` Methode entfernt, da sie `GetSignedPolicy` jetzt dieselbe Funktion erfüllt. Darüber hinaus `GetSignedPolicy` wurde ein dritter Parameter für den Endpunkt Region zugewiesen.

Die **`PutACL`** Methoden **`GetACL`** und

Die `PutACL` Methoden `GetACL` und der Klasse AmazonS3Client wurden als veraltet markiert. Verwenden Sie stattdessen die folgenden neuen Methoden, um auf die Funktionalität dieser Methoden zuzugreifen:`GetBucketACL`,`PutBucketACL`, `GetObjectACL` und. `PutObjectACL`

Veraltete Programmierelemente wurden entfernt

Eine Reihe von Programmierelementen der Amazon S3 S3-Implementierung wurden aus Version 4 des SDK entfernt, darunter Aufzählungswerte, Typen, Methoden, Namespaces usw. Diese sind unten aufgeführt, sofern sie nicht bereits zuvor behandelt wurden, zusammen mit möglichen Schritten, die Sie ergreifen können, um ihre Entfernung zu erleichtern.

- Die `DisableMD5Stream` Immobilie wurde aus der TransferUtilityUploadRequestKlasse entfernt. Verwenden Sie die stattdessen die `DisableDefaultChecksumValidation`-Eigenschaft.

  Außerdem wurde die `CalculateContentMD5Header` Eigenschaft aus der `TransferUtilityUploadRequest` Klasse entfernt. Diese Eigenschaft wird nicht mehr benötigt, da das SDK standardmäßig eine Prüfsumme berechnet.
- Die `ServerSideEncryptionKeyManagementServiceKeyId` Eigenschaften `ServerSideEncryptionMethod` und wurden aus der CopyPartRequestKlasse entfernt. Verwenden Sie stattdessen die Eigenschaften mit den gleichen Namen in der InitiateMultipartUploadRequestKlasse, die in einigen `InitiateMultipartUpload...` Methoden der AmazonS3Client-Klasse verwendet wird.
- Die `Expires` Eigenschaft wurde aus der Klasse entfernt. GetObjectResponse Verwenden Sie die stattdessen die `ExpiresString`-Eigenschaft. Die Zeichenfolge hat möglicherweise kein gültiges Zeitstempelformat, daher sollte Ihr Code die `TryParse` Methode bei der Konvertierung in eine `DateTime` verwenden.
- Veraltete AWS-Region Bezeichner wurden aus der S3Region-Aufzählung entfernt.
- Die `Prefix` Eigenschaft wurde aus der Klasse entfernt. LifecycleRule Verwenden Sie die stattdessen die `Filter`-Eigenschaft.

  Außerdem wurden die `Transition` Eigenschaften `NoncurrentVersionTransition` und aus der `LifecycleRule` Klasse entfernt. Verwenden Sie stattdessen die `Transitions` Sammlungen `NoncurrentVersionTransitions` und.
- Die `Event` Eigenschaft wurde aus der TopicConfigurationKlasse entfernt. Verwenden Sie stattdessen die `Events` Sammlung.
- CalculateContentMD5Header-Eigenschaft. Diese Eigenschaft musste nicht mehr festgelegt werden, da das SDK standardmäßig eine Prüfsumme berechnet.
- Die `Bucket` Eigenschaft wurde aus der SelectObjectContentRequestKlasse entfernt. Verwenden Sie die stattdessen die `BucketName`-Eigenschaft.

- Die `NumberOfUploadThreads` Eigenschaft wurde aus der TransferUtilityConfigKlasse entfernt. Verwenden Sie stattdessen die `ConcurrentServiceRequests` Eigenschaft.

Programmierelemente, die entfernt wurden

Eine Reihe von Programmierelementen wurde aus Version 4 des SDK entfernt, darunter Aufzählungswerte, Typen, Methoden, Namespaces usw. Diese sind unten aufgeführt, sofern sie nicht bereits zuvor behandelt wurden, zusammen mit möglichen Schritten, die Sie ergreifen können, um ihre Entfernung zu erleichtern.

Zum Ansehen von Artikeln öffnen

Der **`Amazon.Auth.AccessControlPolicy.ActionIdentifiers`**-Namespace

Der `Amazon.Auth.AccessControlPolicy.ActionIdentifiers` Namespace wurde entfernt. Dazu gehören IAM-Aktions-IDs, die in der Klasse definiert wurden. `IdentityandAccessManagementActionIdentifiers` Code, der diese Aktionsbezeichner verwendet, sollte so geändert werden, dass er Zeichenkettenwerte des Aktionsnamens verwendet.

Weitere Informationen finden Sie unter the section called “Verwaltete Richtlinien aus JSON erstellen” und im Überblick über JSON-Richtlinien im IAM-Benutzerhandbuch.

Die **`ClientConfig`** Klasse

Die Amazon.Runtime. ClientConfigKlasse ist die Basisklasse von Service-Client-Konfigurationsklassen wie AmazonS3Config. Die folgenden Programmierelemente wurden aus dieser Klasse entfernt.

- Die `DetermineDnsSuffix` Methoden `DetermineServiceURL` und wurden entfernt. Verwenden Sie stattdessen die `DetermineServiceOperationEndpoint` Methode des Service-Clients, zum Beispiel AmazonS3Client. DetermineServiceOperationEndpoint.
- Die `ReadEntireResponse` Immobilie wurde entfernt. Verwenden Sie stattdessen eine der folgenden Optionen:
  - Das `LogResponses` Eigentum von AWSConfigs. LoggingConfigKlasse.
  - Die `LogResponse` Eigenschaft der Client-Konfiguration, zum Beispiel AmazonS3Config.

Der **`Amazon.Runtime`**-Namespace

Der Amazon.Runtime-Namespace wurde wie folgt aktualisiert:

- Die veraltete `ECSTaskCredentials` Klasse wurde aus dem Namespace entfernt. Verwenden Sie stattdessen den GenericContainerCredentialsAnbieter, der auch Amazon EKS Pod Identities unterstützt.
- Die veralteten `StoredProfileCredentials` Klassen `StoredProfileAWSCredentials` und wurden aus dem Namespace entfernt. Verwenden Sie die SDKCredentialsNet-Datei oder die SharedCredentialsFileKlasse der Amazon.Runtime. CredentialManagementStattdessen Namespace.
- Die veraltete `HasCachedAccessTokenAvailable` Methode der SSOAWSCredentialsKlasse wurde aus dem Namespace entfernt.
- Die veraltete `EnvironmentAWSCredentials` Klasse wurde aus dem Namespace entfernt. Verwenden Sie stattdessen die AppConfigAWSCredentials-Klasse.
- Die veraltete `StoredProfileFederatedCredentials` Klasse wurde aus dem Namespace entfernt. Verwenden Sie stattdessen die AWSCredentialsFederated-Klasse.
- Die folgenden veralteten Klassen wurden aus dem Namespace entfernt: `EnvironmentVariableAWSEndpointDiscoveryEnabledProfileAWSEndpointDiscoveryEnable` und. `FallbackEndpointDiscoveryEnabledFactory`
- Die veraltete `UseSigV4` Eigenschaft wurde aus der AmazonWebServiceRequestKlasse entfernt. Verwenden Sie die stattdessen die `SignatureVersion`-Eigenschaft.
- Für die `ProfileIniFile` Klasse im `Amazon.Runtime.Internal.Util` Namespace wird eine überladene Methode aufgerufen. `TryGetSection` Die Versionen der Methode, die den `out` Parameter for nicht unterstützen, `nestedProperties` wurden aus der Klasse entfernt.
- Die veraltete `EventBridgeSigner` Klasse im `Amazon.Runtime.Internal.Auth` Namespace wurde entfernt.
- Die veraltete `Parameters` Wörterbucheigenschaft wurde aus der WebServiceRequestEventArgsKlasse entfernt. Verwenden Sie die stattdessen die `ParameteCollection`-Eigenschaft.

BouncyCastle

Die Quellkopie von BouncyCastle wurde aus Version 4 des SDK entfernt.

Die **`StoredProfileSAMLCredentials`** Klasse

Die veraltete `StoredProfileSAMLCredentials` Klasse im Amazon. SecurityToken.SAML-Namespace wurde entfernt. Verwenden Sie stattdessen die AWSCredentialsFederated-Klasse im Amazon.Runtime-Namespace.

Die **`AWSSDKUtils`** Klasse

Die folgenden Methoden wurden aus der AWSSDKUtilsKlasse entfernt:`ResolveResourcePath`, und. `ProtectEncodedSlashUrlEncode` `ConvertToUnixEpochMilliSeconds`

Die **`ProfileManager`** Klasse

Die veraltete `ProfileManager` Klasse wurde aus dem Amazon.Util-Namespace entfernt. Verwenden Sie die SDKCredentialsNet-Datei oder SharedCredentialsFile-Klasse aus der Amazon.Runtime. CredentialManagementStattdessen einen Namespace.

Die **`AWSConfigs`** Klasse

Die folgenden veralteten Eigenschaften wurden aus der AWSConfigsKlasse entfernt: `LoggingResponseLogging`, und`LogMetrics`. Verwenden Sie die stattdessen die `LoggingConfig`-Eigenschaft.

Die **`ConditionFactory`** Klasse

Die Methode mit der folgenden Signatur wurde aus der ConditionFactoryKlasse entfernt:`NewCondition(ConditionFactory.DateComparisonType, DateTime)`. Verwenden Sie stattdessen die NewConditionUtcMethode.

CloudFront Amazon-Dienstprogramme

Der veraltete `Amazon.CloudFront.Util` Namespace und die veraltete `AmazonCloudFrontUtil` Klasse wurden entfernt.

AWS IoT

In der ListPrincipalThingsResponseKlasse wurde eine veraltete Anpassung für eine `NextToken` Überschreibung zugunsten der Paginierung entfernt.

AWS Lambda

Die folgenden `Invoke...` Methoden der AmazonLambdaClientKlasse wurden entfernt, da die Namen verwirrend waren.

- Die V3-Methode mit der folgenden Signatur wurde entfernt:`InvokeAsyncResponse InvokeAsync(InvokeAsyncRequest)`. Dies ist eine synchrone Methode in V3 des SDK. Verwenden Sie `InvokeResponse Invoke(InvokeRequest)` stattdessen (für synchrone Verarbeitung) oder `Task InvokeAsync(InvokeRequest, CancellationToken)` (für asynchrone Verarbeitung).

- Die V3-Methode mit der folgenden Signatur wurde entfernt:. `Task InvokeAsyncAsync(InvokeAsyncRequest, CancellationToken)` Dies ist eine asynchrone Methode in V3 des SDK. Verwenden Sie stattdessen `Task InvokeAsync(InvokeRequest, CancellationToken)`.

Amazon SageMaker Runtime

Veraltete Konstruktoren für die PayloadPartKlasse wurden entfernt.

Migration von .NET-Standard 1.3

Am 27. Juni 2019 hat Microsoft die Unterstützung für .NET Core 1.0 und .NET Core 1.1 beendet . Nach dieser Ankündigung AWS wurde der Support für .NET Standard 1.3 AWS SDK für .NET am 31. Dezember 2020 eingestellt.

AWS stellte bis zum 1. Oktober 2020 weiterhin Service-Updates und Sicherheitsupdates für das.NET Standard 1.3 bereit. AWS SDK für .NET Nach diesem Datum ging das.NET Standard 1.3-Ziel in den Wartungsmodus über, was bedeutete, dass keine neuen Updates veröffentlicht wurden. Es wurden nur kritische Bugfixes und Sicherheitspatches AWS angewendet.

Am 31. Dezember 2020 AWS SDK für .NET wurde der Support für.NET Standard 1.3 auf dem eingestellt. Nach diesem Datum wurden keine Bugfixes oder Sicherheitspatches mehr angewendet. Artefakte, die mit diesem Ziel erstellt wurden, stehen weiterhin am zum Download zur Verfügung NuGet.

Wichtige Informationen

- Wenn Sie Anwendungen mit .NET Framework ausführen, sind Sie davon nicht betroffen.

- Wenn Sie Anwendungen mit .NET Core 2.0 oder höher ausführen, sind Sie davon nicht betroffen.

- Wenn Sie Anwendungen mit .NET Core 1.0 oder .NET Core 1.1 ausführen, migrieren Sie Ihre Anwendungen auf eine neuere Version von .NET Core, indem Sie die Migrationsanweisungen von Microsoft befolgen. Wir empfehlen mindestens .NET Core 3.1.

- Wenn Sie geschäftskritische Anwendungen ausführen, die derzeit nicht aktualisiert werden können, ist es möglich, weiterhin Ihre aktuelle Version von AWS SDK für .NET zu verwenden.

Bei Fragen oder Bedenken kontaktieren Sie den AWS -Support.

Sicherheit für dieses AWS Produkt oder diesen Service

Cloud-Sicherheit genießt bei Amazon Web Services (AWS) höchste Priorität. Als AWS -Kunde profitieren Sie von einer Rechenzentrums- und Netzwerkarchitektur, die zur Erfüllung der Anforderungen von Organisationen entwickelt wurden, für die Sicherheit eine kritische Bedeutung hat. Sicherheit ist eine gemeinsame Verantwortung zwischen Ihnen AWS und Ihnen. Im Modell der übergreifenden Verantwortlichkeit wird Folgendes mit „Sicherheit der Cloud“ bzw. „Sicherheit in der Cloud“ umschrieben:

Sicherheit der Cloud — AWS ist verantwortlich für den Schutz der Infrastruktur, auf der alle in der AWS Cloud angebotenen Dienste ausgeführt werden, und für die Bereitstellung von Diensten, die Sie sicher nutzen können. Unsere Sicherheitsverantwortung hat bei uns höchste Priorität AWS, und die Wirksamkeit unserer Sicherheit wird im Rahmen der AWS Compliance-Programme regelmäßig von externen Prüfern getestet und verifiziert.

Sicherheit in der Cloud — Ihre Verantwortung richtet sich nach dem von Ihnen genutzten AWS Dienst und anderen Faktoren, wie der Sensibilität Ihrer Daten, den Anforderungen Ihres Unternehmens und den geltenden Gesetzen und Vorschriften.

Dieses AWS Produkt oder dieser Service folgt dem Modell der gemeinsamen Verantwortung in Bezug auf die spezifischen Amazon Web Services (AWS) -Services, die es unterstützt. Informationen zur AWS Servicesicherheit finden Sie auf der Seite mit der Dokumentation zur AWS Servicesicherheit und den AWS Services, für die das AWS Compliance-Programm zur Einhaltung der Vorschriften zuständig ist.

Themen

- Datenschutz in diesem AWS Produkt oder dieser Dienstleistung
- Identitäts- und Zugriffsverwaltung
- Überprüfung der Einhaltung der Vorschriften für dieses AWS Produkt oder diese Dienstleistung
- Ausfallsicherheit für dieses AWS Produkt oder diese Dienstleistung
- Sicherheit der Infrastruktur für dieses AWS Produkt oder diesen Service
- Erzwingen einer TLS-Mindestversion in AWS SDK für .NET
- Migration des Amazon S3 S3-Verschlüsselungsclients

Datenschutz in diesem AWS Produkt oder dieser Dienstleistung

Das AWS Modell der gilt für den Datenschutz in diesem AWS Produkt oder dieser Dienstleistung. Wie in diesem Modell beschrieben, AWS ist verantwortlich für den Schutz der globalen Infrastruktur, auf der alle Systeme laufen AWS Cloud. Sie sind dafür verantwortlich, die Kontrolle über Ihre in dieser Infrastruktur gehosteten Inhalte zu behalten. Sie sind auch für die Sicherheitskonfiguration und die Verwaltungsaufgaben für die von Ihnen verwendeten AWS-Services verantwortlich. Weitere Informationen zum Datenschutz finden Sie unter Häufig gestellte Fragen zum Datenschutz. Informationen zum Datenschutz in Europa finden Sie im Blog-Beitrag AWS -Modell der geteilten Verantwortung und in der DSGVO im AWS -Sicherheitsblog.

Aus Datenschutzgründen empfehlen wir, dass Sie AWS-Konto Anmeldeinformationen schützen und einzelne Benutzer mit AWS IAM Identity Center oder AWS Identity and Access Management (IAM) einrichten. So erhält jeder Benutzer nur die Berechtigungen, die zum Durchführen seiner Aufgaben erforderlich sind. Außerdem empfehlen wir, die Daten mit folgenden Methoden schützen:

- Verwenden Sie für jedes Konto die Multi-Faktor-Authentifizierung (MFA).
- Wird verwendet SSL/TLS , um mit AWS Ressourcen zu kommunizieren. Wir benötigen TLS 1.2 und empfehlen TLS 1.3.
- Richten Sie die API und die Protokollierung von Benutzeraktivitäten mit ein AWS CloudTrail. Informationen zur Verwendung von CloudTrail Pfaden zur Erfassung von AWS Aktivitäten finden Sie unter Arbeiten mit CloudTrail Pfaden im AWS CloudTrail Benutzerhandbuch.
- Verwenden Sie AWS Verschlüsselungslösungen zusammen mit allen darin enthaltenen Standardsicherheitskontrollen AWS-Services.
- Verwenden Sie erweiterte verwaltete Sicherheitsservices wie Amazon Macie, die dabei helfen, in Amazon S3 gespeicherte persönliche Daten zu erkennen und zu schützen.
- Wenn Sie für den Zugriff AWS über eine Befehlszeilenschnittstelle oder eine API FIPS 140-3-validierte kryptografische Module benötigen, verwenden Sie einen FIPS-Endpunkt. Weitere Informationen über verfügbare FIPS-Endpunkte finden Sie unter Federal Information Processing Standard (FIPS) 140-3.

Wir empfehlen dringend, in Freitextfeldern, z. B. im Feld Name, keine vertraulichen oder sensiblen Informationen wie die E-Mail-Adressen Ihrer Kunden einzugeben. Dazu gehört auch, wenn Sie mit diesem AWS Produkt oder Service oder einem anderen AWS-Services über die Konsole, API oder arbeiten. AWS CLI AWS SDKs Alle Daten, die Sie in Tags oder Freitextfelder eingeben, die für Namen verwendet werden, können für Abrechnungs- oder Diagnoseprotokolle verwendet

werden. Wenn Sie eine URL für einen externen Server bereitstellen, empfehlen wir dringend, keine Anmeldeinformationen zur Validierung Ihrer Anforderung an den betreffenden Server in die URL einzuschließen.

Identitäts- und Zugriffsverwaltung

AWS Identity and Access Management (IAM) hilft einem Administrator AWS-Service , den Zugriff auf Ressourcen sicher zu AWS kontrollieren. IAM-Administratoren kontrollieren, wer authentifiziert (angemeldet) und autorisiert werden kann (über Berechtigungen verfügt), um Ressourcen zu verwenden. AWS IAM ist ein Programm AWS-Service , das Sie ohne zusätzliche Kosten nutzen können.

Themen

- Zielgruppe
- Authentifizierung mit Identitäten
- Verwalten des Zugriffs mit Richtlinien
- Wie AWS-Services arbeiten Sie mit IAM
- Fehlerbehebung bei AWS Identität und Zugriff

Zielgruppe

Die Art und Weise, wie Sie AWS Identity and Access Management (IAM) verwenden, hängt von der Arbeit ab, in der Sie tätig sind. AWS

Dienstbenutzer — Wenn Sie dies AWS-Services für Ihre Arbeit verwenden, stellt Ihnen Ihr Administrator die erforderlichen Anmeldeinformationen und Berechtigungen zur Verfügung. Wenn Sie für Ihre Arbeit mehr AWS Funktionen verwenden, benötigen Sie möglicherweise zusätzliche Berechtigungen. Wenn Sie die Funktionsweise der Zugriffskontrolle nachvollziehen, wissen Sie bereits, welche Berechtigungen Sie von Ihrem Administrator anfordern müssen. Falls Sie auf eine Funktion nicht zugreifen können AWS, finden Fehlerbehebung bei AWS Identität und Zugriff Sie weitere Informationen in der Bedienungsanleitung der von AWS-Service Ihnen verwendeten.

Serviceadministrator — Wenn Sie in Ihrem Unternehmen für die AWS Ressourcen verantwortlich sind, haben Sie wahrscheinlich vollen Zugriff auf AWS. Es ist Ihre Aufgabe, zu bestimmen, auf welche AWS Funktionen und Ressourcen Ihre Servicebenutzer zugreifen sollen. Anschließend müssen Sie Anforderungen an Ihren IAM-Administrator senden, um die Berechtigungen der

Servicebenutzer zu ändern. Lesen Sie die Informationen auf dieser Seite, um die Grundkonzepte von IAM nachzuvollziehen. Weitere Informationen darüber, wie Ihr Unternehmen IAM verwenden kann AWS, finden Sie in der Benutzeranleitung des von AWS-Service Ihnen verwendeten.

IAM-Administrator: Wenn Sie als IAM-Administrator fungieren, sollten Sie Einzelheiten dazu kennen, wie Sie Richtlinien zur Verwaltung des Zugriffs auf AWS verfassen können. Beispiele für AWS identitätsbasierte Richtlinien, die Sie in IAM verwenden können, finden Sie im Benutzerhandbuch der AWS-Service von Ihnen verwendeten.

Authentifizierung mit Identitäten

Authentifizierung ist die Art und Weise, wie Sie sich AWS mit Ihren Identitätsdaten anmelden. Sie müssen sich als IAM-Benutzer authentifizieren oder eine IAM-Rolle annehmen. Root-Benutzer des AWS-Kontos

Sie können sich als föderierte Identität anmelden, indem Sie Anmeldeinformationen aus einer Identitätsquelle wie AWS IAM Identity Center (IAM Identity Center), Single Sign-On-Authentifizierung oder Anmeldeinformationen verwenden. Google/Facebook Weitere Informationen zur Anmeldung finden Sie im Benutzerhandbuch unter So melden Sie sich bei Ihrem AWS-Konto anAWS-Anmeldung .

AWS Bietet für den programmatischen Zugriff ein SDK und eine CLI zum kryptografischen Signieren von Anfragen. Weitere Informationen finden Sie im IAM-Benutzerhandbuch unter AWS Signature Version 4 für API-Anfragen.

AWS-Konto Root-Benutzer

Wenn Sie einen erstellen AWS-Konto, beginnen Sie mit einer Anmeldeidentität, dem sogenannten AWS-Konto Root-Benutzer, der vollständigen Zugriff auf alle AWS-Services Ressourcen hat.
Wir empfehlen dringend, den Root-Benutzer nicht für alltägliche Aufgaben zu verwenden.
Informationen zu Aufgaben, für die Root-Benutzeranmeldedaten erforderlich sind, finden Sie im IAM-Benutzerhandbuch unter Aufgaben, für die Root-Benutzeranmeldedaten erforderlich sind.

Verbundidentität

Es hat sich bewährt, dass menschliche Benutzer für den Zugriff AWS-Services mithilfe temporärer Anmeldeinformationen einen Verbund mit einem Identitätsanbieter verwenden müssen.

Eine föderierte Identität ist ein Benutzer aus Ihrem Unternehmensverzeichnis, Ihrem AWS Directory Service Web-Identitätsanbieter oder der AWS-Services mithilfe von Anmeldeinformationen

aus einer Identitätsquelle zugreift. Föderierte Identitäten übernehmen Rollen, die temporäre Anmeldeinformationen bereitstellen.

Für eine zentralisierte Zugriffsverwaltung empfehlen wir. AWS IAM Identity Center Weitere Informationen finden Sie unter Was ist IAM Identity Center? im AWS IAM Identity Center - Benutzerhandbuch.

IAM-Benutzer und -Gruppen

Ein IAM-Benutzer ist eine Identität mit spezifischen Berechtigungen für eine einzelne Person oder Anwendung. Wir empfehlen, temporäre Anmeldeinformationen anstelle von IAM-Benutzern mit langfristigen Anmeldeinformationen zu verwenden. Weitere Informationen finden Sie im IAM-Benutzerhandbuch unter Erfordern, dass menschliche Benutzer den Verbund mit einem Identitätsanbieter verwenden müssen, um AWS mithilfe temporärer Anmeldeinformationen darauf zuzugreifen.

Eine IAM-Gruppe spezifiziert eine Sammlung von IAM-Benutzern und erleichtert die Verwaltung von Berechtigungen für große Benutzergruppen. Weitere Informationen finden Sie im IAM-Benutzerhandbuch unter Anwendungsfälle für IAM-Benutzer.

IAM roles

Eine IAM-Rolle ist eine Identität mit bestimmten Berechtigungen, die temporäre Anmeldeinformationen bereitstellt. Sie können eine Rolle übernehmen, indem Sie von einem Benutzer zu einer IAM-Rolle (Konsole) wechseln oder indem Sie eine AWS CLI oder AWS API-Operation aufrufen. Weitere Informationen finden Sie im IAM-Benutzerhandbuch unter Methoden zur Übernahme einer Rolle.

IAM-Rollen sind nützlich für Verbundbenutzerzugriff, temporäre IAM-Benutzerberechtigungen, kontoübergreifenden Zugriff, dienstübergreifenden Zugriff und Anwendungen, die auf Amazon ausgeführt werden. EC2 Weitere Informationen finden Sie unter Kontoübergreifender Ressourcenzugriff in IAM im IAM-Benutzerhandbuch.

Verwalten des Zugriffs mit Richtlinien

Sie kontrollieren den Zugriff, AWS indem Sie Richtlinien erstellen und diese an Identitäten oder Ressourcen anhängen. AWS Eine Richtlinie definiert Berechtigungen, wenn sie mit einer Identität oder Ressource verknüpft sind. AWS bewertet diese Richtlinien, wenn ein Principal eine Anfrage stellt. Die meisten Richtlinien werden AWS als JSON-Dokumente gespeichert. Weitere Informationen

zu JSON-Richtliniendokumenten finden Sie im IAM-Benutzerhandbuch unter Überblick über JSON-Richtlinien.

Mithilfe von Richtlinien legen Administratoren fest, wer auf was Zugriff hat, indem sie definieren, welcher Principal Aktionen mit welchen Ressourcen und unter welchen Bedingungen ausführen kann.

Standardmäßig haben Benutzer, Gruppen und Rollen keine Berechtigungen. Ein IAM-Administrator erstellt IAM-Richtlinien und fügt sie Rollen hinzu, die Benutzer dann übernehmen können. IAM-Richtlinien definieren Berechtigungen unabhängig von der Methode, die zur Ausführung des Vorgangs verwendet wird.

Identitätsbasierte Richtlinien

Identitätsbasierte Richtlinien sind Richtliniendokumente für JSON-Berechtigungen, die Sie an eine Identität (Benutzer, Gruppe oder Rolle) anhängen. Diese Richtlinien steuern, welche Aktionen Identitäten auf welchen Ressourcen und unter welchen Bedingungen ausführen können. Informationen zum Erstellen identitätsbasierter Richtlinien finden Sie unter Definieren benutzerdefinierter IAM-Berechtigungen mit vom Kunden verwalteten Richtlinien im IAM-Benutzerhandbuch.

Bei identitätsbasierten Richtlinien kann es sich um Inline-Richtlinien (direkt in eine einzelne Identität eingebettet) oder um verwaltete Richtlinien (eigenständige Richtlinien, die mehreren Identitäten zugeordnet sind) handeln. Informationen zur Auswahl zwischen verwalteten und Inline-Richtlinien finden Sie im IAM-Benutzerhandbuch unter Wählen Sie zwischen verwalteten Richtlinien und Inline-Richtlinien.

Ressourcenbasierte Richtlinien

Ressourcenbasierte Richtlinien sind JSON-Richtliniendokumente, die Sie an eine Ressource anfügen. Beispiele hierfür sind Vertrauensrichtlinien für IAM-Rollen und Amazon S3 S3-Bucket-Richtlinien. In Services, die ressourcenbasierte Richtlinien unterstützen, können Service-Administratoren sie verwenden, um den Zugriff auf eine bestimmte Ressource zu steuern. Sie müssen in einer ressourcenbasierten Richtlinie einen Prinzipal angeben.

Ressourcenbasierte Richtlinien sind Richtlinien innerhalb dieses Diensts. Sie können AWS verwaltete Richtlinien von IAM nicht in einer ressourcenbasierten Richtlinie verwenden.

Zugriffskontrolllisten () ACLs

Zugriffskontrolllisten (ACLs) steuern, welche Principals (Kontomitglieder, Benutzer oder Rollen) über Zugriffsberechtigungen für eine Ressource verfügen. ACLs ähneln ressourcenbasierten Richtlinien, verwenden jedoch nicht das JSON-Richtliniendokumentformat.

Amazon S3 und Amazon VPC sind Beispiele für Dienste, die Unterstützung ACLs bieten. AWS WAF Weitere Informationen finden Sie unter Übersicht über ACLs die Zugriffskontrollliste (ACL) im Amazon Simple Storage Service Developer Guide.

Weitere Richtlinientypen

AWS unterstützt zusätzliche Richtlinientypen, mit denen die maximalen Berechtigungen festgelegt werden können, die durch gängigere Richtlinientypen gewährt werden:

- Berechtigungsgrenzen — Legen Sie die maximalen Berechtigungen fest, die eine identitätsbasierte Richtlinie einer IAM-Entität gewähren kann. Weitere Informationen finden Sie unter Berechtigungsgrenzen für IAM-Entitäten im -IAM-Benutzerhandbuch.
- Richtlinien zur Dienststeuerung (SCPs) — Geben Sie die maximalen Berechtigungen für eine Organisation oder Organisationseinheit in an. AWS Organizations Weitere Informationen finden Sie unter Service-Kontrollrichtlinien im AWS Organizations -Benutzerhandbuch.
- Richtlinien zur Ressourcenkontrolle (RCPs) — Legen Sie die maximal verfügbaren Berechtigungen für Ressourcen in Ihren Konten fest. Weitere Informationen finden Sie im AWS Organizations Benutzerhandbuch unter Richtlinien zur Ressourcenkontrolle (RCPs).
- Sitzungsrichtlinien — Erweiterte Richtlinien, die beim Erstellen einer temporären Sitzung für eine Rolle oder einen Verbundbenutzer als Parameter übergeben werden. Weitere Informationen finden Sie unter Sitzungsrichtlinien im IAM-Benutzerhandbuch.

Mehrere Richtlinientypen

Wenn mehrere auf eine Anforderung mehrere Richtlinientypen angewendet werden können, sind die entsprechenden Berechtigungen komplizierter. Informationen darüber, wie AWS bestimmt wird, ob eine Anfrage zulässig ist, wenn mehrere Richtlinientypen betroffen sind, finden Sie unter Bewertungslogik für Richtlinien im IAM-Benutzerhandbuch.

Wie AWS-Services arbeiten Sie mit IAM

Einen allgemeinen Überblick darüber, wie die meisten IAM-Funktionen AWS-Services funktionieren, finden Sie im AWS IAM-Benutzerhandbuch unter Dienste, die mit IAM funktionieren.

Informationen zur Verwendung bestimmter Dienste AWS-Service mit IAM finden Sie im Abschnitt Sicherheit im Benutzerhandbuch des jeweiligen Dienstes.

Fehlerbehebung bei AWS Identität und Zugriff

Verwenden Sie die folgenden Informationen, um häufig auftretende Probleme zu diagnostizieren und zu beheben, die bei der Arbeit mit AWS und IAM auftreten können.

Themen

- Ich bin nicht berechtigt, eine Aktion durchzuführen in AWS
- Ich bin nicht berechtigt, iam auszuführen: PassRole
- Ich möchte Personen außerhalb von mir den Zugriff AWS-Konto auf meine AWS Ressourcen ermöglichen

Ich bin nicht berechtigt, eine Aktion durchzuführen in AWS

Wenn Sie eine Fehlermeldung erhalten, dass Sie nicht zur Durchführung einer Aktion berechtigt sind, müssen Ihre Richtlinien aktualisiert werden, damit Sie die Aktion durchführen können.

Der folgende Beispielfehler tritt auf, wenn der IAM-Benutzer `mateojackson` versucht, über die Konsole Details zu einer fiktiven *`my-example-widget`*-Ressource anzuzeigen, jedoch nicht über `awes:`*`GetWidget`*-Berechtigungen verfügt.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 awes:GetWidget on resource: my-example-widget
```

In diesem Fall muss die Richtlinie für den Benutzer `mateojackson` aktualisiert werden, damit er mit der `awes:`*`GetWidget`*-Aktion auf die *`my-example-widget`*-Ressource zugreifen kann.

Wenn Sie Hilfe benötigen, wenden Sie sich an Ihren AWS Administrator. Ihr Administrator hat Ihnen Ihre Anmeldeinformationen zur Verfügung gestellt.

Ich bin nicht berechtigt, iam auszuführen: PassRole

Wenn Sie eine Fehlermeldung erhalten, dass Sie nicht autorisiert sind, die `iam:PassRole`-Aktion auszuführen, müssen Ihre Richtlinien aktualisiert werden, damit Sie eine Rolle an AWSübergeben können.

Einige AWS-Services ermöglichen es Ihnen, eine bestehende Rolle an diesen Dienst zu übergeben, anstatt eine neue Servicerolle oder eine dienstverknüpfte Rolle zu erstellen. Hierzu benötigen Sie Berechtigungen für die Übergabe der Rolle an den Dienst.

Der folgende Beispielfehler tritt auf, wenn ein IAM-Benutzer mit dem Namen `marymajor` versucht, die Konsole zu verwenden, um eine Aktion in AWS auszuführen. Um die Aktion ausführen zu können, benötigt der Service jedoch Berechtigungen, die von einer Servicerolle gewährt werden. Mary besitzt keine Berechtigungen für die Übergabe der Rolle an den Dienst.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In diesem Fall müssen die Richtlinien von Mary aktualisiert werden, um die Aktion `iam:PassRole` ausführen zu können.

Wenn Sie Hilfe benötigen, wenden Sie sich an Ihren AWS Administrator. Ihr Administrator hat Ihnen Ihre Anmeldeinformationen zur Verfügung gestellt.

Ich möchte Personen außerhalb von mir den Zugriff AWS-Konto auf meine AWS Ressourcen ermöglichen

Sie können eine Rolle erstellen, die Benutzer in anderen Konten oder Personen außerhalb Ihrer Organisation für den Zugriff auf Ihre Ressourcen verwenden können. Sie können festlegen, wem die Übernahme der Rolle anvertraut wird. Für Dienste, die ressourcenbasierte Richtlinien oder Zugriffskontrolllisten (ACLs) unterstützen, können Sie diese Richtlinien verwenden, um Personen Zugriff auf Ihre Ressourcen zu gewähren.

Weitere Informationen finden Sie hier:

- Informationen darüber, ob diese Funktionen AWS unterstützt werden, finden Sie unter. Wie AWS-Services arbeiten Sie mit IAM
- Informationen dazu, wie Sie Zugriff auf Ihre Ressourcen gewähren können, AWS-Konten die Ihnen gehören, finden Sie im IAM-Benutzerhandbuch unter Gewähren des Zugriffs auf einen IAM-Benutzer in einem anderen AWS-Konto , den Sie besitzen.

- Informationen dazu, wie Sie Dritten Zugriff auf Ihre Ressourcen gewähren können AWS-Konten, finden Sie AWS-Konten im IAM-Benutzerhandbuch unter Gewähren des Zugriffs für Dritte.
- Informationen dazu, wie Sie über einen Identitätsverbund Zugriff gewähren, finden Sie unter Gewähren von Zugriff für extern authentifizierte Benutzer (Identitätsverbund) im IAM-Benutzerhandbuch.
- Informationen zum Unterschied zwischen der Verwendung von Rollen und ressourcenbasierten Richtlinien für den kontoübergreifenden Zugriff finden Sie unter Kontoübergreifender Ressourcenzugriff in IAM im IAM-Benutzerhandbuch.

Überprüfung der Einhaltung der Vorschriften für dieses AWS Produkt oder diese Dienstleistung

Informationen darüber, ob AWS-Service ein AWS-Services in den Geltungsbereich bestimmter Compliance-Programme fällt, finden Sie unter Umfang nach Compliance-Programm AWS-Services unter . Wählen Sie dort das Compliance-Programm aus, an dem Sie interessiert sind. Allgemeine Informationen finden Sie unter AWS Compliance-Programme AWS .

Sie können Prüfberichte von Drittanbietern unter herunterladen AWS Artifact. Weitere Informationen finden Sie unter Berichte herunterladen unter .

Ihre Verantwortung für die Einhaltung der Vorschriften bei der Nutzung AWS-Services hängt von der Vertraulichkeit Ihrer Daten, den Compliance-Zielen Ihres Unternehmens und den geltenden Gesetzen und Vorschriften ab. Weitere Informationen zu Ihrer Verantwortung für die Einhaltung der Vorschriften bei der Nutzung AWS-Services finden Sie in der AWS Sicherheitsdokumentation.

Dieses AWS Produkt oder dieser Service folgt dem Modell der gemeinsamen Verantwortung in Bezug auf die spezifischen Amazon Web Services (AWS) -Services, die es unterstützt. Informationen zur AWS Servicesicherheit finden Sie auf der Seite mit der Dokumentation zur AWS Servicesicherheit und den AWS Services, für die das AWS Compliance-Programm zur Einhaltung der Vorschriften zuständig ist.

Ausfallsicherheit für dieses AWS Produkt oder diese Dienstleistung

Die AWS globale Infrastruktur basiert auf AWS-Regionen Availability Zones.

AWS-Regionen bieten mehrere physisch getrennte und isolierte Availability Zones, die über Netzwerke mit niedriger Latenz, hohem Durchsatz und hoher Redundanz miteinander verbunden sind.

Mithilfe von Availability Zones können Sie Anwendungen und Datenbanken erstellen und ausführen, die automatisch Failover zwischen Zonen ausführen, ohne dass es zu Unterbrechungen kommt. Availability Zones sind besser verfügbar, fehlertoleranter und skalierbarer als herkömmliche Infrastrukturen mit einem oder mehreren Rechenzentren.

Weitere Informationen zu AWS Regionen und Availability Zones finden Sie unter AWS Globale Infrastruktur.

Dieses AWS Produkt oder dieser Service folgt dem Modell der gemeinsamen Verantwortung in Bezug auf die spezifischen Amazon Web Services (AWS) -Services, die es unterstützt. Informationen zur AWS Servicesicherheit finden Sie auf der Seite mit der Dokumentation zur AWS Servicesicherheit und den AWS Services, für die das AWS Compliance-Programm zur Einhaltung der Vorschriften zuständig ist.

Sicherheit der Infrastruktur für dieses AWS Produkt oder diesen Service

Dieses AWS Produkt oder dieser Dienst verwendet Managed Services und ist daher durch die AWS globale Netzwerksicherheit geschützt. Informationen zu AWS Sicherheitsdiensten und zum AWS Schutz der Infrastruktur finden Sie unter AWS Cloud-Sicherheit. Informationen zum Entwerfen Ihrer AWS Umgebung unter Verwendung der bewährten Methoden für die Infrastruktursicherheit finden Sie unter Infrastructure Protection in Security Pillar AWS Well-Architected Framework.

Sie verwenden AWS veröffentlichte API-Aufrufe, um über das Netzwerk auf dieses AWS Produkt oder diesen Service zuzugreifen. Kunden müssen Folgendes unterstützen:

- Transport Layer Security (TLS). Wir benötigen TLS 1.2 und empfehlen TLS 1.3.
- Verschlüsselungs-Suiten mit Perfect Forward Secrecy (PFS) wie DHE (Ephemeral Diffie-Hellman) oder ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Die meisten modernen Systeme wie Java 7 und höher unterstützen diese Modi.

Außerdem müssen Anforderungen mit einer Zugriffsschlüssel-ID und einem geheimen Zugriffsschlüssel signiert sein, der einem IAM-Prinzipal zugeordnet ist. Alternativ können Sie mit AWS

Security Token Service (AWS STS) temporäre Sicherheitsanmeldeinformationen erstellen, um die Anforderungen zu signieren.

Dieses AWS Produkt oder dieser Service folgt dem Modell der gemeinsamen Verantwortung in Bezug auf die spezifischen Amazon Web Services (AWS) -Services, die es unterstützt. Informationen zur AWS Servicesicherheit finden Sie auf der Seite mit der Dokumentation zur AWS Servicesicherheit und den AWS Services, für die das AWS Compliance-Programm zur Einhaltung der Vorschriften zuständig ist.

Erzwingen einer TLS-Mindestversion in AWS SDK für .NET

Um die Sicherheit bei der Kommunikation mit AWS Diensten zu erhöhen, sollten Sie den so konfigurieren, AWS SDK für .NET dass er TLS 1.2 oder höher verwendet.

Die bestimmt AWS SDK für .NET anhand der zugrundeliegenden.NET-Laufzeit, welches Sicherheitsprotokoll verwendet werden soll. Standardmäßig verwenden aktuelle Versionen von .NET das zuletzt konfigurierte Protokoll, das vom Betriebssystem unterstützt wird. Ihre Anwendung kann dieses SDK-Verhalten überschreiben, dies wird jedoch nicht empfohlen.

.NET Core

Standardmäßig verwendet .NET Core das zuletzt konfigurierte Protokoll, das vom Betriebssystem unterstützt wird. AWS SDK für .NET bietet keinen Mechanismus, um dies außer Kraft zu setzen.

Wenn Sie eine .NET Core-Version vor 2.1 verwenden, empfehlen wir dringend, dass Sie Ihre .NET Core-Version aktualisieren.

Weitere Informationen zu den einzelnen Betriebssystemen finden Sie im Folgenden.

Windows

Bei modernen Windows-Verteilungen ist die TLS 1.2-Unterstützung standardmäßig aktiviert. Wenn Sie Windows 7 SP1 oder Windows Server 2008 R2 verwenden, müssen Sie sicherstellen SP1, dass die TLS 1.2-Unterstützung in der Registrierung aktiviert ist, wie unter https://learn.microsoft.com/en-us/windows- server/security/tls/tls -registry-settings #tls -12 beschrieben. Wenn Sie eine frühere Verteilung ausführen, müssen Sie Ihr Betriebssystem aktualisieren. Informationen zur Unterstützung von TLS 1.3 in Windows finden Sie in der neuesten Microsoft-Dokumentation für die mindestens erforderlichen Client- oder Serverversionen.

macOS

Wenn Sie .NET Core 2.1 oder höher ausführen, ist TLS 1.2 standardmäßig aktiviert. TLS 1.2 wird von OS X Mavericks 10.9 oder höher unterstützt. .NET Core Version 2.1 und höher erfordern neuere Versionen von macOS, wie unter beschrieben https://learn.microsoft.com/en-us/dotnet/core/install/windows? tabs=net80&pivots=os-macos.

Bei Verwendung von .NET Core 1.0 nutzt .NET Core OpenSSL unter macOS eine Abhängigkeit, die separat installiert werden muss. OpenSSL fügte Unterstützung für TLS 1.2 in Version 1.0.1 und Unterstützung für TLS 1.3 in Version 1.1.1 hinzu.

Linux

.NET Core ist unter Linux OpenSSL erforderlich, das mit vielen Linux-Verteilungen geliefert wird. Aber es kann auch separat installiert werden. OpenSSL fügte Unterstützung für TLS 1.2 in Version 1.0.1 und Unterstützung für TLS 1.3 in Version 1.1.1 hinzu. Wenn Sie eine moderne Version von .NET Core (2.1 oder höher) verwenden und einen Paketmanager installiert haben, wurde wahrscheinlich eine modernere Version von OpenSSL für Sie installiert.

Zur Sicherheit können Sie **`openssl version`** in einem Terminal ausführen und überprüfen, ob die Version höher als 1.0.1 ist.

.NET Framework.

Wenn Sie eine moderne Version von .NET Framework (4.7 oder höher) und eine moderne Version von Windows (mindestens Windows 8 für Clients, Windows Server 2012 oder höher für Server) ausführen, ist TLS 1.2 standardmäßig aktiviert und wird verwendet.

Wenn Sie eine.NET-Framework-Runtime verwenden, die nicht die Betriebssystemeinstellungen (.NET Framework 3.5 bis 4.5.2) verwendet, versucht sie, AWS SDK für .NET die unterstützten Protokolle um Unterstützung für TLS 1.1 und TLS 1.2 zu erweitern. Bei Verwendung von .NET Framework 3.5 ist dies nur erfolgreich, wenn der entsprechende Hotpatch wie folgt installiert wird:

- Windows 10 Version 1511 und Windows Server 2016 — KB3156421
- Windows 8.1 und Windows Server 2012 R2 — KB315452 0
- Windows Server 2012 — KB3154519
- Windows 7 SP1 und Server 2008 R2 SP1 — KB3154518

> **Warning**
>
> Am 15. August 2024 AWS SDK für .NET wurde der Support für.NET Framework 3.5 eingestellt und die Mindestversion von .NET Framework auf 4.7.2 geändert. Weitere Informationen finden Sie im Blogbeitrag Wichtige Änderungen, die demnächst für die Ziele .NET Framework 3.5 und 4.5 von gelten. AWS SDK für .NET

Wenn Ihre Anwendung auf einem neueren.NET Framework unter Windows 7 SP1 oder Windows Server 2008 R2 ausgeführt wird, müssen Sie sicherstellen SP1, dass die TLS 1.2-Unterstützung in der Registrierung aktiviert ist, wie unter https://learn.microsoft.com/en-us/windows- server/security/tls/tls -registry-settings #tls -12 beschrieben. Bei neueren Versionen von Windows ist es standardmäßig aktiviert.

Detaillierte bewährte Methoden für die Verwendung von TLS mit.NET Framework finden Sie im Microsoft-Artikel unter https://learn.microsoft.com/en-us/dotnet/framework/network-programming/tls.

AWS -Tools für PowerShell

AWS -Tools für PowerShellVerwenden Sie die AWS SDK für .NET für alle Aufrufe von AWS Diensten. Das Verhalten Ihrer Umgebung hängt wie folgt von der Version von Windows ab, die PowerShell Sie verwenden.

Windows PowerShell 2.0 bis 5.x

Windows PowerShell 2.0 bis 5.x laufen auf.NET Framework. Mit dem folgenden Befehl können Sie überprüfen, PowerShell von welcher .NET-Laufzeit (2.0 oder 4.0) verwendet wird.

```
$PSVersionTable.CLRVersion
```

- Wenn Sie .NET Runtime 2.0 verwenden, befolgen Sie die obigen Anweisungen in Bezug auf AWS SDK für .NET und .NET Framework 3.5.

  > **Warning**
  >
  > Am 15. August 2024 AWS SDK für .NET wurde der Support für .NET Framework 3.5 eingestellt und die Mindestversion von .NET Framework auf 4.7.2 geändert. Weitere

Informationen finden Sie im Blogbeitrag Wichtige Änderungen, die demnächst für die Ziele .NET Framework 3.5 und 4.5 von gelten. AWS SDK für .NET

- Wenn Sie .NET Runtime 4.0 verwenden, befolgen Sie die obigen Anweisungen in Bezug auf AWS SDK für .NET und .NET Framework 4+.

Windows PowerShell 6.0

Windows PowerShell 6.0 und neuer laufen auf.NET Core. Mit dem folgenden Befehl können Sie überprüfen, welche Version von .NET Core verwendet wird.

```
[System.Reflection.Assembly]::GetEntryAssembly().GetCustomAttributes([System.Runtime.Versioning
 $true).FrameworkName
```

Folgen Sie den zuvor angegebenen Anweisungen für die AWS SDK für .NET und die entsprechende Version von.NET Core.

Xamarin

Für Xamarin finden Sie die Anweisungen unter https://learn.microsoft.com/en-us/ xamarin/cross-platform/app-fundamentals/transport -layer-security. Zusammenfassend:

Für Android

- Android 5.0 oder höher erforderlich.
- Projekteigenschaften, Android-Optionen: Die HttpClient Implementierung muss auf Android und die SSL/TLS Implementierung auf Native TLS 1.2+ eingestellt sein.

Für iOS

- iOS 7 oder höher erforderlich.
- Projekteigenschaften, iOS Build: Die HttpClient Implementierung muss auf NSUrlSitzung eingestellt sein.

Für macOS

- macOS 10.9 oder höher erforderlich.

- Projektoptionen, Build, Mac Build: Die HttpClient Implementierung muss auf NSUrlSitzung eingestellt sein.

Unity

Sie müssen Unity 2018.2 oder höher verwenden und die .NET 4.x gleichwertige Skripting-Laufzeit verwenden. Sie können dies in den Projekteinstellungen, der Konfiguration und dem Player festlegen, wie unter https://docs.unity3d.com/2019.1/Documentation/Manual/ScriptingRuntimeUpgrade.html beschrieben. Die.NET 4.x-Äquivalente Scripting-Laufzeit ermöglicht die Unterstützung von TLS 1.2 für alle Unity-Plattformen, auf denen Mono oder IL2 CPP ausgeführt wird.

Browser (für Blazor) WebAssembly

WebAssembly läuft im Browser statt auf dem Server und verwendet den Browser für die Verarbeitung von HTTP-Verkehr. Daher wird die TLS-Unterstützung durch die Browser-Unterstützung bestimmt.

Blazor WebAssembly, in der Vorschauversion für ASP.NET Core 3.1, wird nur in Browsern unterstützt, die dies unterstützen WebAssembly, wie unter -platforms beschrieben. https://learn.microsoft.com/en-us/ aspnet/core/blazor/supported Alle gängigen Browser unterstützten TLS 1.2 vor der Unterstützung. WebAssembly Wenn dies für Ihren Browser der Fall ist, kann Ihre App bei Ausführung über TLS 1.2 kommunizieren.

Weitere Informationen und Verifizierung finden Sie in der Dokumentation Ihres Browsers.

Migration des Amazon S3 S3-Verschlüsselungsclients

In diesem Thema erfahren Sie, wie Sie Ihre Anwendungen von Version 1 (V1) des Amazon Simple Storage Service (Amazon S3) -Verschlüsselungsclients auf Version 2 (V2) migrieren und die Anwendungsverfügbarkeit während des gesamten Migrationsprozesses sicherstellen.

Objekte, die mit dem V2-Client verschlüsselt wurden, können mit dem V1-Client nicht entschlüsselt werden. Um die Migration zum neuen Client zu vereinfachen, ohne alle Objekte auf einmal erneut verschlüsseln zu müssen, wurde ein „V1-Transitional“ -Client bereitgestellt. Dieser Client kann sowohl V1- als auch V2-verschlüsselte Objekte entschlüsseln, verschlüsselt Objekte jedoch nur im V1-kompatiblen Format. Der V2-Client kann sowohl V1- als auch V2-verschlüsselte Objekte entschlüsseln (sofern er für V1-Objekte aktiviert ist), verschlüsselt Objekte jedoch nur im V2-kompatiblen Format.

Überblick über die Migration

Diese Migration erfolgt in drei Phasen. Diese Phasen werden hier vorgestellt und später ausführlich beschrieben. Jede Phase muss für alle Clients abgeschlossen sein, die gemeinsam genutzte Objekte verwenden, bevor die nächste Phase gestartet wird.

1. Aktualisieren Sie bestehende Clients auf V1-Transition-Clients, um neue Formate lesen zu können. Aktualisieren Sie zunächst Ihre Anwendungen so, dass sie vom V1-Transitiony-Client statt vom V1-Client abhängig sind. Der V1-Transitiony-Client ermöglicht es Ihrem vorhandenen Code, Objekte zu entschlüsseln, die von den neuen V2-Clients geschrieben wurden, und Objekte, die in einem V1-kompatiblen Format geschrieben wurden.

   > **Note**
   >
   > Der V1-Transitiony-Client wird nur für Migrationszwecke bereitgestellt. Fahren Sie mit dem Upgrade auf den V2-Client fort, nachdem Sie zum V1-Transitiony-Client gewechselt haben.

2. Migrieren Sie V1-Transition-Clients auf V2-Clients, um neue Formate zu schreiben. Ersetzen Sie als Nächstes alle V1-Transition-Clients in Ihren Anwendungen durch V2-Clients und setzen Sie das Sicherheitsprofil auf. `V2AndLegacy` Wenn Sie dieses Sicherheitsprofil auf V2-Clients einrichten, können diese Clients Objekte entschlüsseln, die im V1-kompatiblen Format verschlüsselt wurden.
3. Aktualisieren Sie V2-Clients so, dass sie V1-Formate nicht mehr lesen. Nachdem alle Clients auf V2 migriert wurden und alle Objekte im V2-kompatiblen Format verschlüsselt oder neu verschlüsselt wurden, stellen Sie das V2-Sicherheitsprofil schließlich auf statt auf ein. `V2` `V2AndLegacy` Dadurch wird die Entschlüsselung von Objekten im V1-kompatiblen Format verhindert.

Aktualisieren Sie bestehende Clients auf V1-Transitional-Clients, um neue Formate lesen zu können

Der V2-Verschlüsselungsclient verwendet Verschlüsselungsalgorithmen, die ältere Versionen des Clients nicht unterstützen. Der erste Schritt der Migration besteht darin, Ihre V1-Entschlüsselungsclients zu aktualisieren, damit sie das neue Format lesen können.

Mit dem V1-Transitional-Client können Ihre Anwendungen sowohl V1- als auch V2-verschlüsselte Objekte entschlüsseln. Dieser Client ist Teil des Amazon.Extensions.S3.Encryption-Pakets. NuGet Führen Sie die folgenden Schritte für jede Ihrer Anwendungen aus, um den V1-Transitiony-Client zu verwenden.

1. Gehen Sie von einer neuen Abhängigkeit vom Paket Amazon.Extensions.S3.Encryption aus. Wenn Ihr Projekt direkt vom .S3 abhängt oder. AWSSDK AWSSDK KeyManagementServicePakete, Sie müssen diese Abhängigkeiten entweder aktualisieren oder entfernen, damit ihre aktualisierten Versionen mit diesem neuen Paket übernommen werden.
2. Ändern Sie die entsprechende `using` Anweisung wie folgt von `Amazon.S3.Encryption` bis`Amazon.Extensions.S3.Encryption`:

   ```
   // using Amazon.S3.Encryption;
     using Amazon.Extensions.S3.Encryption;
   ```

3. Erstellen Sie Ihre Anwendung neu und stellen Sie sie erneut bereit.

Der V1-Transitiony-Client ist vollständig API-kompatibel mit dem V1-Client, sodass keine weiteren Codeänderungen erforderlich sind.

Migrieren Sie V1-Transitiony-Clients zu V2-Clients, um neue Formate zu schreiben

Der V2-Client ist Teil des Amazon.Extensions.S3.Encryption-Pakets NuGet . Er ermöglicht es Ihren Anwendungen, sowohl V1- als auch V2-verschlüsselte Objekte zu entschlüsseln (sofern entsprechend konfiguriert), verschlüsselt Objekte jedoch nur im V2-kompatiblen Format.

Nachdem Sie Ihre vorhandenen Clients so aktualisiert haben, dass sie das neue Verschlüsselungsformat lesen, können Sie damit fortfahren, Ihre Anwendungen sicher auf die V2-Verschlüsselungs- und Entschlüsselungsclients zu aktualisieren. Führen Sie für jede Ihrer Anwendungen die folgenden Schritte aus, um den V2-Client zu verwenden:

1. Ändern Sie `EncryptionMaterials` zu `EncryptionMaterialsV2`.
   a. Bei Verwendung von KMS:
      i. Geben Sie eine KMS-Schlüssel-ID an.

ii. Deklarieren Sie die Verschlüsselungsmethode, die Sie verwenden; das heißt,`KmsType.KmsContext`.

iii. Stellen Sie KMS einen Verschlüsselungskontext zur Verfügung, der mit diesem Datenschlüssel verknüpft werden soll. Sie können ein leeres Wörterbuch senden (der Amazon-Verschlüsselungskontext wird trotzdem zusammengeführt), aber die Angabe von zusätzlichem Kontext ist erwünscht.

b. Wenn Sie vom Benutzer bereitgestellte Methoden zum Umbrechen von Schlüsseln verwenden (symmetrische oder asymmetrische Verschlüsselung):

i. Stellen Sie eine `AES` oder eine `RSA` Instanz bereit, die die Verschlüsselungsmaterialien enthält.

ii. Deklarieren Sie, welcher Verschlüsselungsalgorithmus verwendet werden soll, d. `SymmetricAlgorithmType.AesGcm` h. oder`AsymmetricAlgorithmType.RsaOaepSha1`.

2. Wechseln Sie `AmazonS3CryptoConfiguration` zu, `AmazonS3CryptoConfigurationV2` wobei die `SecurityProfile` Eigenschaft auf gesetzt ist`SecurityProfile.V2AndLegacy`.
3. Ändern Sie `AmazonS3EncryptionClient` zu `AmazonS3EncryptionClientV2`. Dieser Client übernimmt die neu konvertierten `EncryptionMaterialsV2` Objekte `AmazonS3CryptoConfigurationV2` und Objekte aus den vorherigen Schritten.

Beispiel: KMS zu KMS+Kontext

Vor der Migration

```
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);
```

Nach der Migration

```
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);
```

Beispiel: Symmetrischer Algorithmus (AES-CBC zu AES-GCM Key Wrap)

`StorageMode` kann `ObjectMetadata` oder `InstructionFile` sein.

Vor der Migration

```
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);
```

Nach der Migration

```
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**
>
> Lesen Sie bei der Entschlüsselung mit AES-GCM das gesamte Objekt bis zum Ende, bevor Sie die entschlüsselten Daten verwenden. Dadurch wird überprüft, ob das Objekt seit der Verschlüsselung nicht geändert wurde.

Beispiel: Asymmetrischer Algorithmus (RSA to RSA-OAEP-SHA 1 Key Wrap)

`StorageMode` kann `ObjectMetadata` oder `InstructionFile` sein.

Vor der Migration

```
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);
```

Nach der Migration

```
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);
```

Aktualisieren Sie V2-Clients so, dass sie V1-Formate nicht mehr lesen

Irgendwann werden alle Objekte mit einem V2-Client verschlüsselt oder neu verschlüsselt worden sein. Nach Abschluss dieser Konvertierung können Sie die V1-Kompatibilität in den V2-Clients deaktivieren, indem Sie die `SecurityProfile` Eigenschaft auf setzen`SecurityProfile.V2`, wie im folgenden Codeausschnitt gezeigt.

```
//var configuration = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2AndLegacy);
var configuration = new AmazonS3CryptoConfigurationV2(SecurityProfile.V2);
```

Besondere Überlegungen für die AWS SDK für .NET

Dieser Abschnitt enthält Überlegungen für spezielle Fälle, in denen die normalen Konfigurationen oder Verfahren nicht angemessen oder ausreichend sind.

Themen

- Beschaffung von Baugruppen für AWS SDK für .NET
- Zugreifen auf Anmeldeinformationen und Profile in einer Anwendung
- Besondere Überlegungen zur Unity-Unterstützung
- Besondere Überlegungen zur Xamarin-Unterstützung

Beschaffung von Baugruppen für AWS SDK für .NET

In diesem Thema wird beschrieben, wie Sie die AWSSDK Assemblys abrufen und lokal (oder lokal) speichern können, um sie in Ihren Projekten zu verwenden. Dies ist nicht die empfohlene Methode für den Umgang mit SDK-Verweisen, ist aber in einigen Umgebungen erforderlich.

> **Note**
>
> Die empfohlene Methode für den Umgang mit SDK-Referenzen besteht darin, nur die NuGet Pakete herunterzuladen und zu installieren, die jedes Projekt benötigt. Diese Methode wird unter beschrieben AWSSDK Pakete installieren mit NuGet.

Wenn Sie NuGet Pakete nicht pro Projekt herunterladen und installieren können oder dürfen, stehen Ihnen die folgenden Optionen zur Verfügung.

Laden Sie ZIP-Dateien herunter und entpacken Sie sie

(Denken Sie daran, dass dies nicht die empfohlene Methode für den Umgang mit Verweisen auf die ist AWS SDK für .NET.)

1. Laden Sie eine der folgenden ZIP-Dateien herunter:

   - aws-sdk-net8.0.zip — Assemblys, die .NET 8 und höher unterstützen.
   - aws-sdk-netcoreapp3.1.zip — Assemblys, die .NET Core 3.1 und höher unterstützen.

- aws-sdk-netstandard2.0.zip — Assemblys, die .NET Standard 2.0 und 2.1 unterstützen.
- aws-sdk-net472.zip — Assemblys, die .NET Framework 4.5 und höher unterstützen.

2. Extrahieren Sie die Assemblys in einen „Download“ -Ordner auf Ihrem Dateisystem; es spielt keine Rolle, wo. Notieren Sie sich diesen Ordner.
3. Wenn Sie Ihr Projekt einrichten, erhalten Sie die erforderlichen Assemblys aus diesem Ordner, wie unter beschriebenInstallieren Sie AWSSDK Baugruppen ohne NuGet.

Zugreifen auf Anmeldeinformationen und Profile in einer Anwendung

Die bevorzugte Methode für die Verwendung von Anmeldeinformationen besteht darin, es dem AWS SDK für .NET zu ermöglichen, sie für Sie zu finden und abzurufen, wie unter beschriebenAuflösung von Anmeldeinformationen und Profilen.

Sie können Ihre Anwendung jedoch auch so konfigurieren, dass sie aktiv Profile und Anmeldeinformationen abruft und diese Anmeldeinformationen dann explizit beim Erstellen eines AWS Service-Clients verwendet.

Verwenden Sie Klassen aus der Amazon.Runtime, um Profile und Anmeldeinformationen aktiv abzurufen. CredentialManagementNamespace.

- Verwenden Sie die SharedCredentialsFileKlasse, um ein Profil in einer Datei zu finden, die AWS das Dateiformat für AWS Anmeldeinformationen verwendet (entweder die Datei mit gemeinsamen Anmeldeinformationen an ihrem Standardspeicherort oder eine Datei mit benutzerdefinierten Anmeldeinformationen). Dateien in diesem Format werden in diesem Text der Kürze halber manchmal einfach als Anmeldeinformationsdateien bezeichnet.
- Verwenden Sie die Net SDKCredentials File-Klasse, um ein Profil im SDK Store zu finden.
- Verwenden Sie je nach Konfiguration einer Klasseneigenschaft die Klasse, um sowohl in einer Anmeldeinformationsdatei als auch im CredentialProfileStoreChainSDK-Speicher zu suchen.

  Sie können diese Klasse verwenden, um Profile zu finden. Sie können diese Klasse auch verwenden, um AWS Anmeldeinformationen direkt anzufordern, anstatt die `AWSCredentialsFactory` Klasse zu verwenden (siehe unten).
- Verwenden Sie die AWSCredentialsFactory-Klasse, um verschiedene Arten von Anmeldeinformationen aus einem Profil abzurufen oder zu erstellen.

Die folgenden Abschnitte enthalten Beispiele für diese Klassen.

Beispiele für den Unterricht CredentialProfileStoreChain

Sie können Anmeldeinformationen oder Profile von der CredentialProfileStoreChainKlasse abrufen, indem Sie die TryGetProfileMethoden TryGetAWSCredentialsoder verwenden. Die `ProfilesLocation` Eigenschaft der Klasse bestimmt das Verhalten der Methoden wie folgt:

- Wenn der Wert Null oder leer `ProfilesLocation` ist, durchsuchen Sie den SDK-Speicher, sofern die Plattform dies unterstützt, und suchen Sie dann im Standardverzeichnis nach der Datei mit den gemeinsamen AWS Anmeldeinformationen.
- Wenn die `ProfilesLocation` Eigenschaft einen Wert enthält, suchen Sie in der Datei mit den Anmeldeinformationen, die in der Eigenschaft angegeben sind.

Rufen Sie die Anmeldeinformationen aus dem SDK-Store oder der Datei mit den gemeinsam genutzten AWS Anmeldeinformationen ab

Dieses Beispiel zeigt Ihnen, wie Sie mithilfe der `CredentialProfileStoreChain` Klasse Anmeldeinformationen abrufen und diese Anmeldeinformationen dann verwenden, um ein AmazonS3Client-Objekt zu erstellen. Die Anmeldeinformationen können aus dem SDK-Store oder aus der Datei mit gemeinsam genutzten AWS Anmeldeinformationen am Standardspeicherort stammen.

In diesem Beispiel wird auch Amazon.Runtime verwendet. AWSCredentialsKlasse.

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

Holen Sie sich ein Profil aus dem SDK Store oder der Datei mit den gemeinsam genutzten AWS Anmeldeinformationen

Dieses Beispiel zeigt Ihnen, wie Sie mithilfe der CredentialProfileStoreChain Klasse ein Profil abrufen. Die Anmeldeinformationen können aus dem SDK-Speicher oder aus der Datei mit gemeinsam genutzten AWS Anmeldeinformationen am Standardspeicherort stammen.

In diesem Beispiel wird auch die CredentialProfileKlasse verwendet.

```
var chain = new CredentialProfileStoreChain();
CredentialProfile basicProfile;
if (chain.TryGetProfile("basic_profile", out basicProfile))
{
    // Use basicProfile
}
```

Ruft Anmeldeinformationen aus einer benutzerdefinierten Anmeldeinformationsdatei ab

Dieses Beispiel zeigt Ihnen, wie Sie mithilfe der CredentialProfileStoreChain Klasse Anmeldeinformationen abrufen können. Die Anmeldeinformationen stammen aus einer Datei, die das AWS Anmeldeinformat-Dateiformat verwendet, sich aber an einem anderen Speicherort befindet.

In diesem Beispiel wird auch Amazon.Runtime verwendet. AWSCredentialsKlasse.

```
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
}
```

Beispiele für Klassen SharedCredentialsFile und AWSCredentials Factory

Erstellen Sie einen AmazonS3Client mithilfe der Klasse SharedCredentialsFile

Dieses Beispiel zeigt Ihnen, wie Sie ein Profil in der Datei mit gemeinsam genutzten AWS Anmeldeinformationen suchen, AWS Anmeldeinformationen aus dem Profil erstellen und dann die Anmeldeinformationen verwenden, um ein AmazonS3Client-Objekt zu erstellen. Das Beispiel verwendet die Klasse. SharedCredentialsFile

In diesem Beispiel werden auch die CredentialProfileKlasse und die Amazon.Runtime verwendet. AWSCredentialsKlasse.

```
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**
>
> Die Net SDKCredentials File-Klasse kann auf genau die gleiche Weise verwendet werden, außer dass Sie ein neues Net SDKCredentials File-Objekt anstelle eines SharedCredentialsFile Objekts instanziieren würden.

Besondere Überlegungen zur Unity-Unterstützung

Wenn Sie den.NET Standard 2.0 für Ihre Unity-Anwendung verwenden, muss Ihre Anwendung direkt auf die AWS SDK für .NET Assemblys (DLL-Dateien) verweisen, anstatt sie zu verwenden NuGet. AWS SDK für .NET Angesichts dieser Anforderung müssen Sie die folgenden wichtigen Aktionen ausführen.

- Sie müssen sich die AWS SDK für .NET Baugruppen besorgen und sie auf Ihr Projekt anwenden. Informationen dazu, wie Sie dies tun können, finden Sie Laden Sie ZIP-Dateien herunter und entpacken Sie sie im ThemaBeschaffung von AWSSDK Baugruppen.

- Sie müssen neben DLLs for AWSSDK.Core und den anderen AWS Diensten, die Sie verwenden, Folgendes DLLs in Ihr Unity-Projekt aufnehmen. Ab Version 3.5.109 von enthält die AWS SDK für .NET.NET-Standard-ZIP-Datei diese zusätzlichen. DLLs
  - Microsoft.Bcl. AsyncInterfaces.dll
  - System.Laufzeit. CompilerServices.Unsafe.dll
  - System.Threading.Tasks.Extensions.dll

- Wenn Sie IL2CPP verwenden, um Ihr Unity-Projekt zu erstellen, müssen Sie Ihrem Asset-Ordner eine `link.xml` Datei hinzufügen, um Code-Stripping zu verhindern. Die `link.xml` Datei muss alle AWSSDK Assemblys auflisten, die Sie verwenden, und jede muss das `preserve="all"` Attribut enthalten. Der folgende Ausschnitt zeigt ein Beispiel für diese Datei.

```
<linker>
    <assembly fullname="AWSSDK.Core" preserve="all"/>
    <assembly fullname="AWSSDK.DynamoDBv2" preserve="all"/>
    <assembly fullname="AWSSDK.Lambda" preserve="all"/>
</linker>
```

> **Note**
>
> Interessante Hintergrundinformationen zu dieser Anforderung finden Sie im Artikel unter https://aws.amazon.com/blogs/developer/ referencing-the-aws-sdk - for-net-standard -2-0- - uwp/. from-unity-xamarin-or

Besondere Überlegungen zur Xamarin-Unterstützung

Xamarin-Projekte (neu und vorhanden) müssen auf .NET Standard 2.0 ausgerichtet werden. Weitere Informationen finden Sie unter Standard .NET 2.0-Unterstützung in Xamarin.Forms und .NET-Implementierungsunterstützung.

Lesen Sie auch die Informationen zu Portable Class Library und Xamarin.

API-Referenz für AWS SDK für .NET

Die AWS SDK für .NET stellt eine API bereit, mit der Sie auf AWS Dienste zugreifen können. Welche Klassen und Methoden in der API verfügbar sind, finden Sie in der AWS SDK für .NET API-Referenz.

Zusätzlich zu der oben angegebenen allgemeinen Referenz enthält jedes der Beispiele unter Geführte Codebeispiele diesem Abschnitt Verweise auf die spezifischen Methoden und Klassen, die in diesem Beispiel verwendet werden.

Über API-Referenzversionen

Die zuvor beschriebene API-Referenz bezieht sich auf Version 3.0 und höher von AWS SDK für .NET.

Informationen zur Migration von älteren Versionen des SDK finden Sie unter Migrieren Sie Ihr Projekt

Informationen zu veralteten Inhalten für frühere Versionen der SDK-API-Referenz finden Sie in den folgenden Artikeln:

- AWS SDK für .NET API-Referenz V1 (veraltet)
- AWS SDK für .NET API-Referenz V2 (veraltet)

Um eine veraltete AWS SDK für .NET API-Referenz anzuzeigen, müssen Sie sie extrahieren und einen Webbrowser konfigurieren. Die folgenden Anweisungen sind Beispiele dafür, wie Sie dies tun können. Sie basieren auf der Verwendung des Google Chrome-Webbrowsers auf einem Windows-System. Passen Sie sie an Ihren spezifischen Webbrowser und Ihr Betriebssystem an.

Sehen Sie sich die veraltete API-Referenz V1 an

1. Laden Sie die ZIP-Datei für die veraltete AWS SDK für .NET API-Referenz V1 herunter. Standardmäßig lautet der Name der ZIP-Datei. `sdkfornet-api-ref_v1_deprecated.zip` Dieser Name wird in diesen Anweisungen durchgängig verwendet.
2. Platzieren Sie die ZIP-Datei in einem Ordner Ihrer Wahl. Für diese Anweisungen wird davon ausgegangen, dass der Ordnername lautet`C:\work\temp\api-refs\V1`.

3. Klicken Sie mit der rechten Maustaste auf die ZIP-Datei und wählen Sie Alle extrahieren. Akzeptieren Sie den Standardspeicherort `C:\work\temp\api-refs\V1\sdkfornet-api-ref_v1` für diese Anweisungen.
4. Erstellen Sie im `C:\work\temp\api-refs\V1` Ordner eine Verknüpfung für Google Chrome. Achten Sie darauf, die ursprüngliche Anwendung nicht zu verschieben.
5. Legen Sie in den Eigenschaften der neuen Verknüpfung die folgenden Felder fest:

   - Ziel: `"<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"`

     Verwenden Sie für das `--user-data-dir` Argument einen Ordnernamen, der für Ihre Umgebung geeignet ist. Der Ordner muss nicht existieren.
   - Fangen Sie an in: `C:\work\temp\api-refs\V1`
6. Geben Sie der Verknüpfung einen passenden Namen.
7. Öffnen Sie die Verknüpfung, um die alte API-Referenz anzuzeigen.

Sehen Sie sich die veraltete API-Referenz V2 an

1. Laden Sie die ZIP-Datei für die veraltete AWS SDK für .NET API-Referenz V2 herunter. Standardmäßig lautet der Name der ZIP-Datei. `sdkfornet-api-ref_v2_deprecated.zip` Dieser Name wird in diesen Anweisungen durchgängig verwendet.
2. Platzieren Sie die ZIP-Datei in einem Ordner Ihrer Wahl. Für diese Anweisungen wird davon ausgegangen, dass der Ordnername lautet`C:\work\temp\api-refs\V2`.
3. Klicken Sie mit der rechten Maustaste auf die ZIP-Datei und wählen Sie Alle extrahieren. Akzeptieren Sie den Standardspeicherort `C:\work\temp\api-refs\V2\sdkfornet-api-ref_v2` für diese Anweisungen.
4. Erstellen Sie im `C:\work\temp\api-refs\V2` Ordner eine Verknüpfung für Google Chrome. Achten Sie darauf, die ursprüngliche Anwendung nicht zu verschieben.
5. Legen Sie in den Eigenschaften der neuen Verknüpfung die folgenden Felder fest:

   - Ziel: `"<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"`

Verwenden Sie für das `--user-data-dir` Argument einen Ordnernamen, der für Ihre Umgebung geeignet ist. Der Ordner muss nicht existieren.

- Fangen Sie an in: `C:\work\temp\api-refs\V2`

6. Geben Sie der Verknüpfung einen passenden Namen.
7. Öffnen Sie die Verknüpfung, um die alte API-Referenz anzuzeigen.

Dokumentverlauf

In der folgenden Tabelle werden die wichtigen Änderungen seit der letzten Version des AWS SDK für .NET Developer Guide beschrieben. Für Benachrichtigungen über Aktualisierungen dieser Dokumentation können Sie einen RSS-Feed abonnieren.

Änderung	Beschreibung	Datum
Was ist neu	Ein Hinweis zu Version 2.0 des AWS Deploy Tools for .NET CLI wurde hinzugefügt.	17. Oktober 2025
AWS OpsWorks	AWS OpsWorks hat das Lebensende erreicht. Es wurde aus der Dokumentation entfernt.	16. September 2025
Was ist neu	End-of-support wurde für Version 3 des angekündigt AWS SDK für .NET.	21. August 2025
Was ist neu	Allgemeine Verfügbarkeit für den Distributed Cache Provider	2. Juli 2025
Framework für die Nachrichtenverarbeitung	Das AWS Message Processing Framework für.NET ist jetzt allgemein verfügbar!	5. Mai 2025
Was ist neu	Version 4 von AWS SDK für .NET ist allgemein verfügbar! Informationen zur Migration Ihrer Anwendungen auf Version 4 finden Sie unter Migration zu Version 4.	28. April 2025

Migration auf Version 4 von AWS SDK für .NET	Enthält Vorschauinformationen zur Migration von Anwendungen, die mit dem erstellt wurden AWS SDK für .NET , auf Version 4.	03. März 2025
Integration AWS mit .NET Aspire in der AWS SDK für .NET	Informationen darüber, wie AWS die Integration in .NET Aspire erfolgte, finden Sie in der AWS SDK für .NET.	15. Februar 2025
Beobachtbarkeit	Kündigte die GA-Version aus Gründen der Beobachtbarkeit an	10. Februar 2025
Was ist neu	Es wurden Informationen über das neue Standardverhalten für den Integritätsschutz hinzugefügt.	15. Januar 2025
Was ist neu	Es wurden Informationen zur vierten Vorschauversion der AWS SDK für .NET Version 4 hinzugefügt.	15. November 2024